frontiers in Neuroscience

ORIGINAL RESEARCH
published: 12 May 2020
doi: 10.3389/fnins.2020.00406

Mixed-Precision Deep Learning Based on Computational Memory

S. R. Nandakumar[1], Manuel Le Gallo[1]*, Christophe Piveteau[1,2], Vinay Joshi[1,3], Giovanni Mariani[1], Irem Boybat[1,4], Geethan Karunaratne[1,2], Riduan Khaddam-Aljameh[1,2], Urs Egger[1], Anastasios Petropoulos[1,5], Theodore Antonakopoulos[5], Bipin Rajendran[3]*, Abu Sebastian[1]* and Evangelos Eleftheriou[1]

[1] IBM Research - Zurich, Rüschlikon, Switzerland, [2] Department of Information Technology and Electrical Engineering, ETH Zurich, Zurich, Switzerland, [3] Engineering Department, King's College London, London, United Kingdom, [4] Ecole Polytechnique Federale de Lausanne (EPFL), Institute of Electrical Engineering, Lausanne, Switzerland, [5] Department of Electrical and Computers Engineering, University of Patras, Rio Achaia, Greece

OPEN ACCESS

Edited by:
Gert Cauwenberghs,
University of California, San Diego,
United States

Reviewed by:
Priyadarshini Panda,
Yale University, United States
Duygu Kuzum,
University of California, San Diego,
United States
Matthew Marinella,
Sandia National Laboratories (SNL),
United States

***Correspondence:**
Manuel Le Gallo
anu@zurich.ibm.com
Bipin Rajendran
bipin.rajendran@kcl.ac.uk
Abu Sebastian
ase@zurich.ibm.com

Specialty section:
This article was submitted to
Neuromorphic Engineering,
a section of the journal
Frontiers in Neuroscience

Received: 11 December 2019
Accepted: 03 April 2020
Published: 12 May 2020

Citation:
Nandakumar SR, Le Gallo M, Piveteau C, Joshi V, Mariani G, Boybat I, Karunaratne G, Khaddam-Aljameh R, Egger U, Petropoulos A, Antonakopoulos T, Rajendran B, Sebastian A and Eleftheriou E (2020) Mixed-Precision Deep Learning Based on Computational Memory. Front. Neurosci. 14:406. doi: 10.3389/fnins.2020.00406

Deep neural networks (DNNs) have revolutionized the field of artificial intelligence and have achieved unprecedented success in cognitive tasks such as image and speech recognition. Training of large DNNs, however, is computationally intensive and this has motivated the search for novel computing architectures targeting this application. A computational memory unit with nanoscale resistive memory devices organized in crossbar arrays could store the synaptic weights in their conductance states and perform the expensive weighted summations in place in a non-von Neumann manner. However, updating the conductance states in a reliable manner during the weight update process is a fundamental challenge that limits the training accuracy of such an implementation. Here, we propose a mixed-precision architecture that combines a computational memory unit performing the weighted summations and imprecise conductance updates with a digital processing unit that accumulates the weight updates in high precision. A combined hardware/software training experiment of a multilayer perceptron based on the proposed architecture using a phase-change memory (PCM) array achieves 97.73% test accuracy on the task of classifying handwritten digits (based on the MNIST dataset), within 0.6% of the software baseline. The architecture is further evaluated using accurate behavioral models of PCM on a wide class of networks, namely convolutional neural networks, long-short-term-memory networks, and generative-adversarial networks. Accuracies comparable to those of floating-point implementations are achieved without being constrained by the non-idealities associated with the PCM devices. A system-level study demonstrates 172× improvement in energy efficiency of the architecture when used for training a multilayer perceptron compared with a dedicated fully digital 32-bit implementation.

Keywords: phase-change memory, in-memory computing, deep learning, mixed-signal design, memristive devices

1. INTRODUCTION

Loosely inspired by the adaptive parallel computing architecture of the brain, deep neural networks (DNNs) consist of layers of neurons and weighted interconnections called synapses. These synaptic weights can be learned using known real-world examples to perform a given task on new unknown data. Gradient descent based algorithms for training DNNs have been successful in achieving human-like accuracy in several cognitive tasks. The training typically involves three stages. During forward propagation, training data is propagated through the DNN to determine the network response. The final neuron layer responses are compared with the desired outputs to compute the resulting error. The objective of the training process is to reduce this error by minimizing a cost function. During backward propagation, the error is propagated throughout the network layers to determine the gradients of the cost function with respect to all the weights. During the weight update stage, the weights are updated based on the gradient information. This sequence is repeated several times over the entire dataset, making training a computationally intensive task (LeCun et al., 2015). Furthermore, when training is performed on conventional von Neumann computing systems that store the large weight matrices in off-chip memory, constant shuttling of data between memory and processor occurs. These aspects make the training of large DNNs very time-consuming, in spite of the availability of high-performance computing resources such as general purpose graphical processing units (GPGPUs). Also, the high-power consumption of this training approach is prohibitive for its widespread application in emerging domains such as the internet of things and edge computing, motivating the search for new architectures for deep learning.

In-memory computing is a non-von Neumann concept that makes use of the physical attributes of memory devices organized in a computational memory unit to perform computations in-place (Ielmini and Wong, 2018; Sebastian et al., 2020). Recent demonstrations include the execution of bulk bit-wise operations (Seshadri et al., 2016), detection of temporal correlations (Sebastian et al., 2017), and matrix-vector multiplications (Hu et al., 2016; Burr et al., 2017; Sheridan et al., 2017; Le Gallo et al., 2018a,b; Li et al., 2018b). The matrix-vector multiplications can be performed in constant computational time complexity using crossbar arrays of resistive memory (memristive) devices (Wong and Salahuddin, 2015; Hu et al., 2016; Ielmini and Wong, 2018). If the network weights are stored as the conductance states of the memristive devices at the crosspoints, then the weighted summations (or matrix-vector multiplications) necessary during the data-propagation stages (forward and backward) of training DNNs can be performed in-place using the computational memory, with significantly reduced data movement (Burr et al., 2015; Prezioso et al., 2015; Gokmen and Vlasov, 2016; Yao et al., 2017; Li et al., 2018a; Sun et al., 2018). However, realizing accurate and gradual modulations of the conductance of the memristive devices for the weight update stage has posed a major challenge in utilizing computational memory to achieve accurate DNN training (Yu, 2018).

The conductance modifications based on atomic rearrangement in nanoscale memristive devices are stochastic, non-linear, and asymmetric as well as of limited granularity (Wouters et al., 2015; Nandakumar et al., 2018a). This has led to significantly reduced classification accuracies compared with software baselines in training experiments using existing memristive devices (Burr et al., 2015). There have been several proposals to improve the precision of synaptic devices. A multi-memristive architecture uses multiple devices per synapse and programs one of them chosen based on a global selector during weight update (Boybat et al., 2018a). Another approach, which uses multiple devices per synapse, further improves the precision by assigning significance to the devices as in a positional number system such as base two. The smaller updates are accumulated in the least significant synaptic device and periodically carried over to higher significant analog memory devices accurately (Agarwal et al., 2017). Hence, all devices in the array must be reprogrammed every time the carry is performed, which brings additional time and energy overheads. A similar approach uses a 3T1C (3 transistor 1 capacitor) cell to accumulate smaller updates and transfer them to PCM periodically using closed-loop iterative programming (Ambrogio et al., 2018). So far, the precision offered by these more complex and expensive synaptic architectures has only been sufficient to demonstrate software-equivalent accuracies in end-to-end training of multi-layer perceptrons for MNIST image classification. All these approaches use a parallel weight update scheme by sending overlapping pulses from the rows and columns, thereby implementing an approximate outer product and potentially updating all the devices in the array in parallel. Each outer product needs to be applied to the arrays one at a time (either after every training example or one by one after a batch of examples), leading to a large number of pulses applied to the devices. This has significant ramifications for device endurance, and the requirements on the number of conductance states to achieve accurate training (Gokmen and Vlasov, 2016; Yu, 2018). Hence, this weight update scheme is best suited for fully-connected networks trained one sample at a time and is limited to training with stochastic gradient descent without momentum, which is a severe constraint on its applicability to a wide range of DNNs. The use of convolution layers, weight updates based on a mini-batch of samples as opposed to a single example, optimizers such as ADAM (Kingma and Ba, 2015), and techniques such as batch normalization (Ioffe and Szegedy, 2015) have been crucial for achieving high learning accuracy in recent DNNs.

Meanwhile, there is a significant body of work in the conventional digital domain using reduced precision arithmetic for accelerating DNN training (Courbariaux et al., 2015; Gupta et al., 2015; Merolla et al., 2016; Hubara et al., 2017; Zhang et al., 2017). Recent studies show that it is possible to reduce the precision of the weights used in the multiply-accumulate operations (during the forward and backward propagations) to even 1 bit, as long as the weight gradients are accumulated in high-precision (Courbariaux et al., 2015). This indicates the possibility of accelerating DNN training using programmable low-precision computational memory, provided that we address the challenge of reliably maintaining the high-precision gradient information. Designing the optimizer in the digital domain rather than in the analog domain permits the implementation of complex learning schemes that can be supported by general-purpose computing systems, as well as maintaining the high-precision in the gradient accumulation, which is necessary to be as high as 32-bit for training state-of-the-art networks (Micikevicius et al., 2018). However, in contrast to the fully digital mixed-precision architectures which uses statistically accurate rounding operations to convert high-precision weights to low precision weights and subsequently make use of error-free digital computation, the weight updates in analog memory devices using programming pulses are highly inaccurate and stochastic. Moreover, the weights stored in the computational memory are affected by noise and temporal conductance variations. Hence, it is not evident if the digital mixed-precision approach translates successfully to a computational memory based deep learning architecture.

Building on these insights, we present a mixed-precision computational memory architecture (MCA) to train DNNs. First, we experimentally demonstrate the efficacy of the architecture to deliver performance close to equivalent floating-point simulations on the task of classifying handwritten digits from the MNIST dataset. Subsequently, we validate the approach through simulations to train a convolutional neural network (CNN) on the CIFAR-10 dataset, a long-short-term-memory (LSTM) network on the Penn Treebank dataset, and a generative-adversarial network (GAN) to generate MNIST digits.

2. RESULTS

2.1. Mixed-Precision Computational Memory Architecture

A schematic illustration of the MCA for training DNNs is shown in **Figure 1**. It consists of a computational memory unit comprising several memristive crossbar arrays, and a high-precision digital computing unit. If the weights W_{ji} in any layer of a DNN (**Figure 1A**) are mapped to the device conductance values G_{ji} in the computational memory with an optional scaling factor, then the desired weighted summation operation during the data-propagation stages of DNN training can be implemented as follows. For the forward propagation, the neuron activations, x_i, are converted to voltages, V_{x_i}, and applied to the crossbar rows. Currents will flow through individual devices based on their conductance and the total current through any column, $I_j = \Sigma_i G_{ji} V_{x_i}$, will correspond to $\Sigma_i W_{ji} x_i$, that becomes the input for the next neuron layer. Similarly, for the backward propagation through the same layer, the voltages V_{δ_j} corresponding to the error δ_j are applied to the columns of the same crossbar array and the weighted sum obtained along the rows, $\Sigma_j W_{ji} \delta_j$, can be used to determine the error δ_i of the preceding layer.

The desired weight updates are determined as $\Delta W_{ji} = \eta \delta_j x_i$, where η is the learning rate. We accumulate these updates in a variable χ in the high-precision digital unit. The accumulated weight updates are transferred to the devices by applying single-shot programming pulses, without using an iterative write-verify scheme. Let ϵ denote the average conductance change that can be reliably programmed into the devices in the computation memory unit using a given pulse. Then, the number of programming pulses p to be applied can be determined by rounding χ/ϵ toward zero. The programming pulses are chosen to increase or decrease the device conductance depending on the sign of p, and χ is decremented by $p\epsilon$ after programming. Effectively, we are transferring the accumulated weight update to the device when it becomes comparable to the device programming granularity. Note that the conductances are updated by applying programming pulses blindly without correcting for the difference between the desired and observed conductance change. In spite of this, the achievable accuracy of

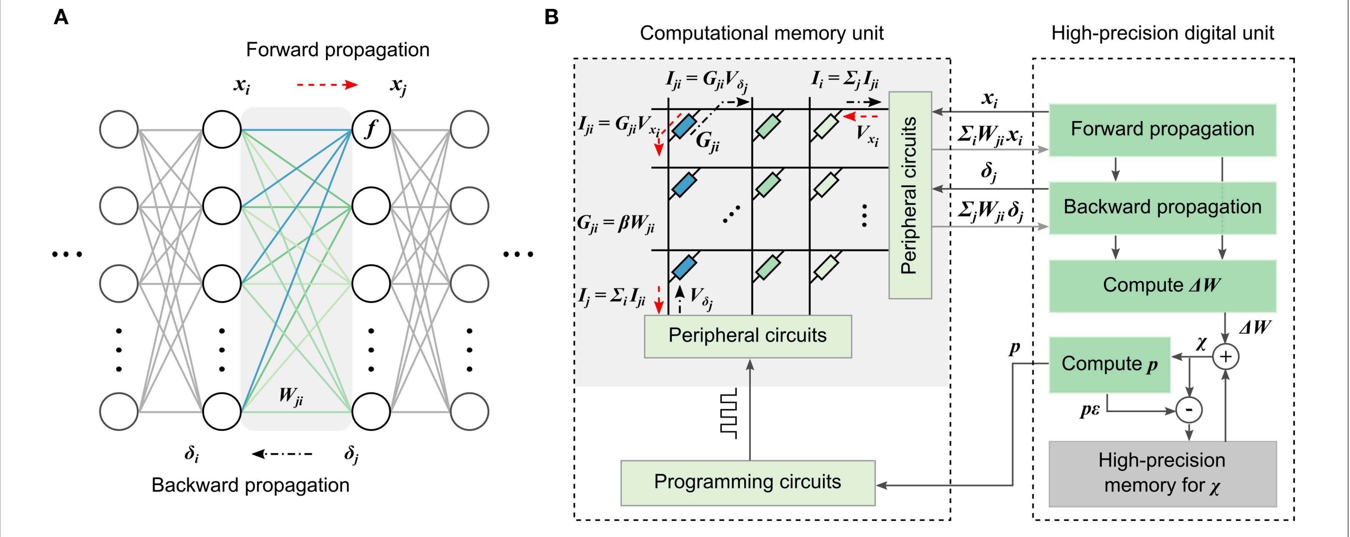

FIGURE 1 | Mixed-precision computational memory architecture for deep learning. **(A)** A neural network consisting of layers of neurons with weighted interconnects. During forward propagation, the neuron response, x_i, is weighted according to the connection strengths, W_{ji}, and summed. Subsequently, a non-linear function, f, is applied to determine the next neuron layer response, x_j. During backward propagation, the error, δ_j, is back-propagated though the weight layer connections, W_{ji}, to determine the error, δ_i, of the preceding layer. **(B)** The mixed-precision architecture consisting of a computational memory unit and a high-precision digital unit. The computational memory unit has several crossbar arrays whose device conductance values G_{ji} represent the weights W_{ji} of the DNN layers. The crossbar arrays perform the weighted summations during the forward and backward propagations. The resulting x and δ values are used to determine the weight updates, ΔW, in the digital unit. The ΔW values are accumulated in the variable, χ. The conductance values are updated using $p = \lfloor \chi/\epsilon \rfloor$ number of pulses applied to the corresponding devices in the computational memory unit, where ϵ represents the device update granularity.

DNNs trained with MCA is extremely robust to the nonlinearity, stochasticity, and asymmetry of conductance changes originating from existing nanoscale memristive devices (Nandakumar et al., 2018b).

2.2. Characterization and Modeling of PCM Devices

Phase-change memory (PCM) devices are used to realize the computational memory for the experimental validation of MCA. PCM is arguably the most advanced memristive technology that has found applications in the space of storage-class memory (Burr et al., 2016) and novel computing paradigms such as neuromorphic computing (Kuzum et al., 2011; Tuma et al., 2016; Sebastian et al., 2018; Joshi et al., 2020) and computational memory (Cassinerio et al., 2013; Sebastian et al., 2017; Le Gallo et al., 2018a). A PCM device consists of a nanometric volume of a chalcogenide phase-change alloy sandwiched between two electrodes. The phase-change material is in the crystalline phase in an as-fabricated device. By applying a current pulse of sufficient amplitude (typically referred to as the RESET pulse) an amorphous region around the narrow bottom electrode is created via melt-quench process. The resulting "mushroom-type" phase configuration is schematically shown in **Figure 2A**. The device will be in a high resistance state if the amorphous region blocks the conductance path between the two electrodes. This amorphous region can be partially crystallized by a SET pulse that heats the device (via Joule heating) to its crystallization temperature regime (Sebastian et al., 2014). With the successive application of such SET pulses, there is a progressive increase in the device conductance. This analog storage capability and the accumulative behavior arising from the crystallization dynamics are central to the application of PCM in training DNNs.

We employ a prototype chip fabricated in 90 nm CMOS technology integrating an array of doped $Ge_2Sb_2Te_5$ (GST) PCM devices (see section A.1). To characterize the gradual conductance evolution in PCM, 10,000 devices are initialized to a distribution around 0.06 μS and are programmed with a sequence of 20 SET pulses of amplitude 90 μA and duration 50 ns. The conductance changes show significant randomness, which is attributed to the inherent stochasticity associated with the crystallization process (Le Gallo et al., 2016), together with device-to-device variability (Tuma et al., 2016; Boybat et al., 2018a). The statistics of cumulative conductance evolution are shown in **Figure 2A**. The conductance evolves in a state-dependent manner and tends to saturate with the number of programming pulses, hence exhibiting a nonlinear accumulative behavior. We analyzed the conductance evolution due to the SET pulses, and developed a comprehensive statistical model capturing the accumulative behavior that shows remarkable

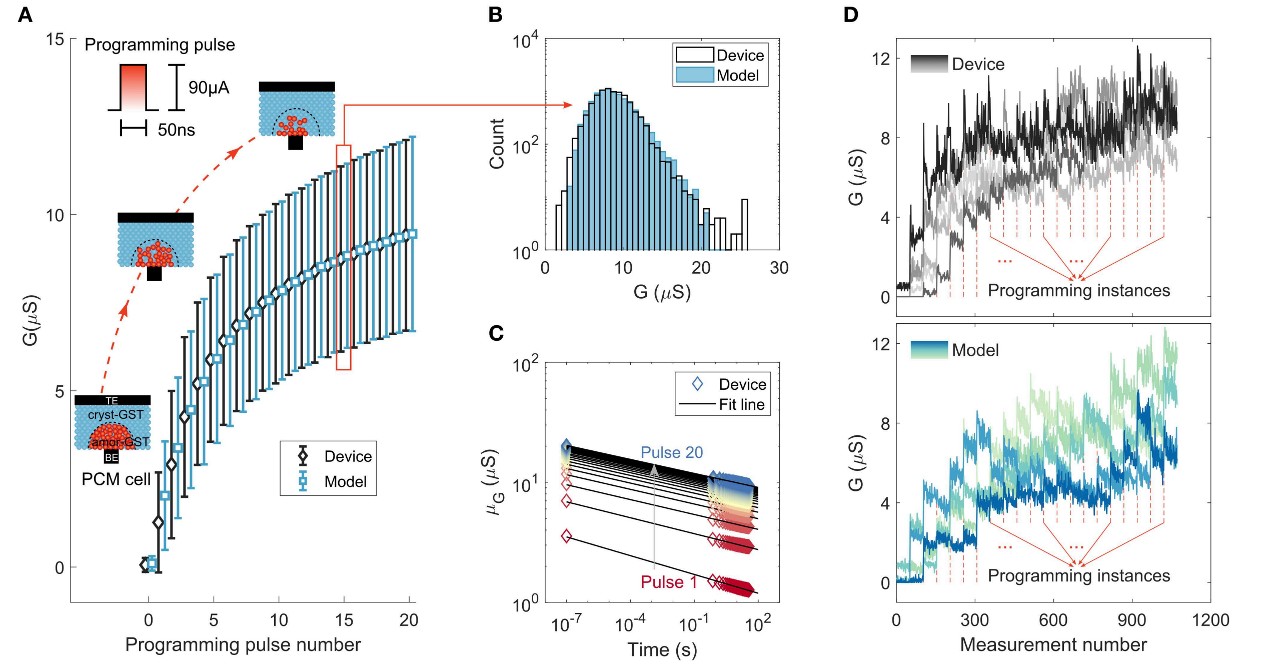

FIGURE 2 | Phase-change memory characterization experiments and model response. **(A)** The mean and standard deviation of device conductance values (and the corresponding model response) as a function of the number of SET pulses of 90 μA amplitude and 50 ns duration. The conductance was read 38.6 s after the application of each SET pulse. The 10,000 PCM devices used for the measurement were initialized to a distribution around 0.06 μS. **(B)** The distribution of conductance values compared to that predicted by the model after the application of 15 SET pulses. **(C)** The average conductance drift of the states programmed after each SET pulse. The corresponding model fit is based on the relation, $G(t) = G(t_0)(t/t_0)^{-\nu}$, that relates the conductance G after time t from programming to the conductance measurement at time t_0 and drift exponent ν. Each color corresponds to the conductance read after the application of a certain number of SET pulses, ranging from 1 to 20. **(D)** Experimentally measured conductance evolution from 5 devices upon application of successive SET pulses compared to that predicted by the model. These measurements are based on 50 reads that follow each of the 20 programming instances. Each line with different color shade corresponds to a different device.

agreement with the measured data (Nandakumar et al., 2018c) (see **Figures 2A,B** and **Supplementary Note 1**). Note that, the conductance response curve is unidirectional and hence asymmetric as we cannot achieve a progressive decrease in the conductance values with the application of successive RESET pulses of the same amplitude.

The devices also exhibit a drift behavior attributed to the structural relaxation of the melt-quenched amorphous phase (Le Gallo et al., 2018). The mean conductance evolution after each programming event as a function of time is plotted in **Figure 2C**. Surprisingly, we find that the drift re-initiates every time a SET pulse is applied (Nandakumar et al., 2018c) (see **Supplementary Note 1**), which could be attributed to the creation of a new unstable glass state due to the atomic rearrangement that is triggered by the application of each SET pulse. In addition to the conductance drift, there are also significant fluctuations in the conductance values (read noise) mostly arising from the $1/f$ noise exhibited by amorphous phase-change materials (Nardone et al., 2009). The statistical model response from a few instances incorporating the programming non-linearity, stochasticity, drift, and instantaneous read noise along with actual device measurements are shown in **Figure 2D**, indicating the similar trend in conductance evolution between the model and the experiment at an individual device level.

2.3. Training Experiment for Handwritten Digit Classification

We experimentally demonstrate the efficacy of the MCA by training a two-layer perceptron to perform handwritten digit classification (**Figure 3A**). Each weight of the network, W, is realized using two PCM devices in a differential configuration ($W \propto (G_p - G_n)$). The 198,760 weights in the network are mapped to 397,520 PCM devices in the hardware platform (see section A.2). The network is trained using 60,000 training images from the MNIST dataset for 30 epochs. The devices are initialized to a conductance distribution with mean 1.6 μS and standard deviation of 0.83 μS. These device conductance values are read from hardware, scaled to the network weights, and used for the data-propagation stages. The resulting weight updates are accumulated in the variable χ. When the magnitude of χ exceeds ϵ ($= 0.096$, corresponding to an average conductance change of 0.77 μS per programming pulse), a 50 ns pulse with an amplitude of 90 μA is applied to G_p to increase the weight if $\chi > 0$ or to G_n to decrease the weight if $\chi < 0$; $|\chi|$ is then reduced by ϵ. These device updates are performed using blind single-shot pulses without a read-verify operation and the device states are not used to determine the number or shape of programming pulses. Since the continuous SET programming could cause some of the devices to saturate during

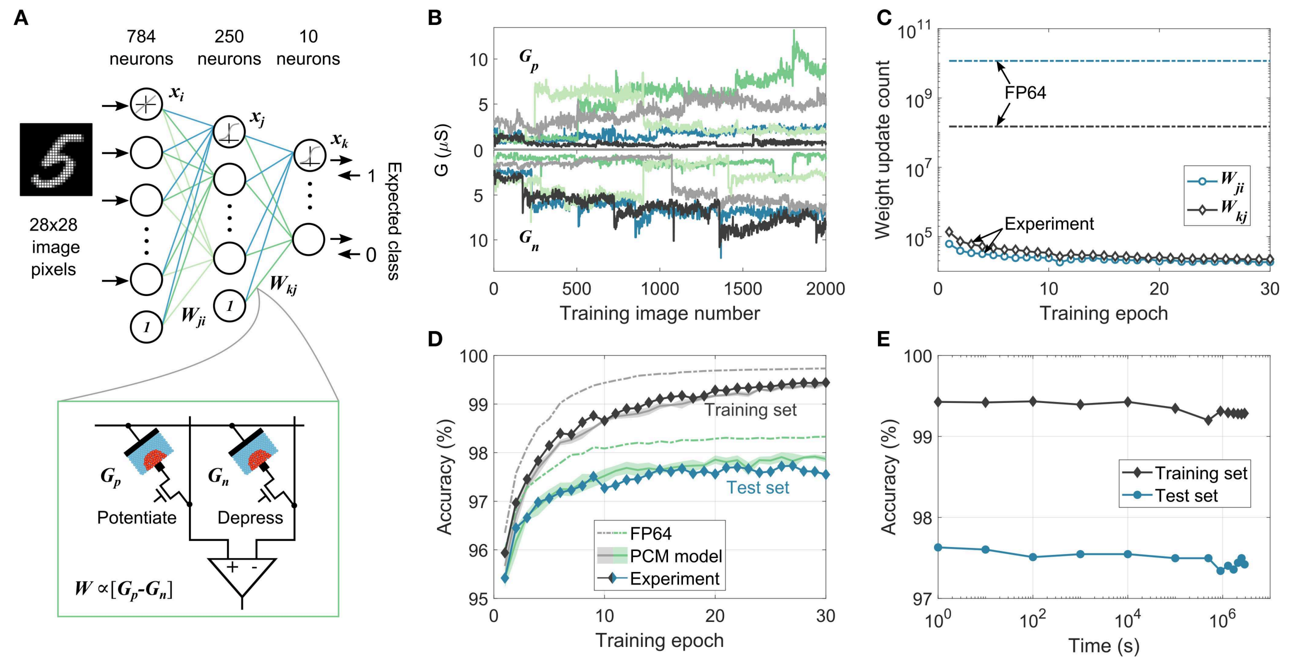

FIGURE 3 | MCA training experiment using on-chip PCM devices for handwritten digit classification. **(A)** Network structure used for the on-chip mixed-precision training experiment for MNIST data classification. Each weight, W, in the network is realized as the difference in conductance values of two PCM devices, G_p and G_n. **(B)** Stochastic conductance evolution during training of G_p and G_n values corresponding to 5 arbitrarily chosen synaptic weights from the second layer. Each color corresponds to a different synaptic weight. **(C)** The number of device updates per epoch from the two weight layers in mixed-precision training experiment and high-precision software training (FP64), showing the highly sparse nature of weight update in MCA. **(D)** Classification accuracies on the training (dark-shaded curves) and test (blue-shaded curves) set from the mixed-precision training experiment. The maximum experimental test set accuracy, 97.73%, is within 0.57% of that obtained in the FP64 training. The experimental behavior is closely matched by the training simulation using the PCM model. The shaded areas in the PCM model curves represent one standard deviation over 5 training simulations. **(E)** Inference performed using the trained PCM weights on-chip on the training and test dataset as a function of time elapsed after training showing negligible accuracy drop over a period of 1 month.

training, a weight refresh operation is performed every 100 training images to detect and reprogram the saturated synapses. After each training example involving a device update, all the devices in the second layer and 785 pairs of devices from the first layer are read along with the updated conductance values to use for the subsequent data-propagation step (see section A.2.2). A separate validation experiment confirms that near identical results are obtained when the read voltage is varied in accordance with the neuron activations and error vectors for every matrix-vector multiplication during training and testing (see **Supplementary Note 2**). The resulting evolution of conductance pairs, G_p and G_n, for five arbitrarily chosen synapses from the second layer is shown in **Figure 3B**. It illustrates the stochastic conductance update, drift between multiple training images, and the read noise experienced by the neural network during training. Also, due to the accumulate-and-program nature of the mixed-precision training, only a few devices are updated after each image. In **Figure 3C**, the number of weight updates per epoch in each layer during training is shown. Compared to the high-precision training where all the weights are updated after each image, there are more than three orders of magnitude reduction in the number of updates in the mixed-precision scheme, thereby reducing the device programming overhead.

At the end of each training epoch, all the PCM conductance values are read from the array and are used to evaluate the classification performance of the network on the entire training set and on a disjoint set of 10,000 test images (**Figure 3D**). The network achieved a maximum test accuracy of 97.73%, only 0.57% lower than the equivalent classification accuracy of 98.30% achieved in the high-precision training. In comparison, applying directly the weight updates to the analog PCM devices without accumulating them in digital with the MCA results in a maximum test accuracy of only 83% in simulation (Nandakumar et al., 2018c). The high-precision comparable training performance achieved by the MCA, where the computational memory comprises noisy non-linear devices with highly stochastic behavior, demonstrates the existence of a solution to these complex deep learning problems in the device-generated weight space. And even more remarkably, it highlights the ability of the MCA to successfully find such solutions. We used the PCM model to validate the training experiment using simulations and the resulting training and test accuracies are plotted in **Figure 3D**. The model was able to predict the experimental classification accuracies on both the training and the test sets within 0.3 %, making it a valuable tool to evaluate the trainability of PCM-based computational memory for more complex deep learning applications. The model was also able to predict the distribution of synaptic weights across the two layers remarkably well (see **Supplementary Note 3**). It also indicated that the accuracy drop from the high-precision baseline training observed in the experiment is mostly attributed to PCM programming stochasticity (see **Supplementary Note 4**). After training, the network weights in the PCM array were read repeatedly over time and the classification performance (inference) was evaluated (**Figure 3E**). It can be seen that the classification accuracy drops by a mere 0.3 % over a time period exceeding a month. This clearly illustrates the feasibility of using trained PCM based computational memory as an inference engine (see section A.2.3).

The use of PCM devices in a differential configuration necessitates the refresh operation. Even though experiments show that the training methodology is robust to a range of refresh schemes (see **Supplementary Note 5**), it does lead to additional complexity. But remarkably, the mixed-precision scheme can deal with even highly asymmetric conductance responses such as in the case where a single PCM device is used to represent the synaptic weights. We performed such an experiment realizing potentiation via SET pulses while depression was achieved using a RESET pulse. To achieve a bipolar weight distribution, a reference conductance level was introduced (see section A.2.4). By using different values of ϵ for potentiation and depression, a maximum test accuracy of 97.47% was achieved within 30 training epochs (see **Supplementary Figure 1**). Even if the RESET pulse causes the PCM to be programmed to a certain low conductance value regardless of its initial conductance, using an average value of this conductance transition to represent ϵ for depression was sufficient to obtain satisfactory training performance. These results conclusively show the efficacy of the mixed-precision training approach and provide a pathway to overcome the stringent requirements on the device update precision and symmetry hitherto thought to be necessary to achieve high performance from memristive device based learning systems (Burr et al., 2015; Gokmen and Vlasov, 2016; Kim et al., 2017; Ambrogio et al., 2018).

2.4. Training Simulations of Larger Networks

The applicability of the MCA training approach to a wider class of problems is verified by performing simulation studies based on the PCM model described in section 2.2. The simulator is implemented as an extension to the TensorFlow deep learning framework. Custom TensorFlow operations are implemented that take into account the various attributes of the PCM devices such as stochastic and nonlinear conductance update, read noise, and conductance drift as well as the characteristics of the data converters (see section A.3.1 and **Supplementary Note 6**). This simulator is used to evaluate the efficacy of the MCA on three networks: a CNN for classifying CIFAR-10 dataset images, an LSTM network for character level language modeling, and a GAN for image synthesis based on the MNIST dataset.

CNNs have become a central tool for many domains in computer vision and also for other applications such as audio analysis in the frequency domain. Their power stems from the translation-invariant weight sharing that significantly reduces the number of parameters needed to extract relevant features from images. As shown in **Figure 4A**, the investigated network consists of three sets of two convolution layers with ReLU (rectified linear unit) activation followed by max-pooling and dropout, and three fully-connected layers at the end. This network has approximately 1.5 million trainable parameters in total (see section A.3.2 and **Supplementary Note 7**). The convolution layer weights are mapped to the PCM devices of the computational memory crossbar arrays by unrolling the filter weights and stacking them to form a 2D matrix (Gokmen et al.,

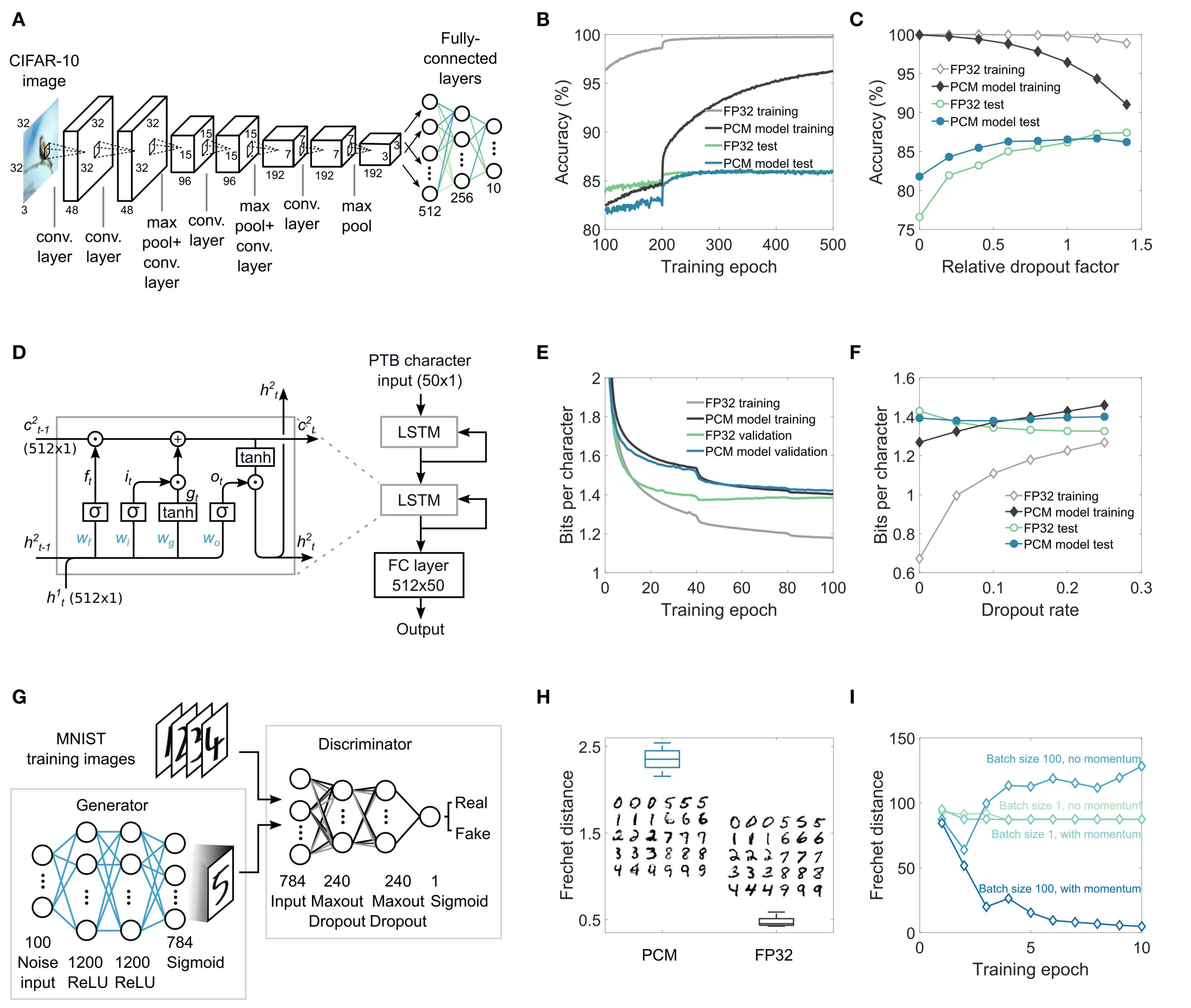

FIGURE 4 | MCA training validation on complex networks. **(A)** Convolutional neural network for image classification on CIFAR-10 dataset used for MCA training simulations. The two convolution layers followed by maxpooling and dropout are repeated thrice, and are followed by three fully-connected layers. **(B)** The classification performance on the training and the test datasets during training. It can be seen that the test accuracy corresponding to MCA-based training eventually exceeds that from the high-precision (FP32) training. **(C)** The maximal training and test accuracies obtained as a function of the dropout rates. In the absence of dropout, MCA-based training significantly outperforms FP32-based training. **(D)** The LSTM network used for MCA training simulations. Two LSTM cells with 512 hidden units followed by a fully-connected (FC) layer are used. **(E)** The BPC as a function of the epoch number on training and validation sets shows that after 100 epochs, the validation BPC is comparable between the MCA and FP32 approaches. The network uses a dropout rate of 0.15 between non-recurring connections. **(F)** The best BPC obtained after training as a function of the dropout rate indicates that without any dropout, MCA-based training delivers better test performance (lower BPC) than FP32 training. **(G)** The GAN network used for MCA training simulations. The generator and discriminator networks are fully-connected. The discriminator and the generator are trained intermittently using real images from the MNIST dataset and the generated images from the generator. **(H)** The Frechet distance obtained from MCA training and the generated images are compared to that obtained from the FP32 training. **(I)** The performance measured in terms of the Frechet distance as a function of the number of epochs obtained for different mini-batch sizes and optimizers for FP32 training. To obtain convergence, mini-batch size >1 and the use of momentum are necessary in the training of the GAN.

2017). The network is trained on the CIFAR-10 classification benchmark dataset. The training and test classification accuracies as a function of training epochs are shown in **Figure 4B**. The maximal test accuracy of the network trained via MCA (86.46 ± 0.25%) is similar to that obtained from equivalent high-precision training using 32-bit floating-point precision (FP32) (86.24 ± 0.19%). However, this is achieved while having a significantly lower training accuracy for MCA, which is suggestive of some beneficial regularization effects arising from the use of stochastic PCM devices to represent synaptic weights. To understand this regularization effect further, we investigated the maximal training and test accuracies as a function of the dropout rates (see

Figure 4C) (see section A.3.3 for more details). It was found that the optimal dropout rate for the network trained via MCA is lower than that for the network trained in FP32. Indeed, if the dropout rate is tuned properly, the test accuracy of the network trained in FP32 could marginally outperform that of the one trained with MCA. Without any dropout, the MCA-trained network outperforms the one trained via FP32 which suffers from significant overfitting.

LSTM networks are a class of recurrent neural networks used mainly for language modeling and temporal sequence learning. The LSTM cells are a natural fit for crossbar arrays, as they basically consist of fully-connected layers (Li et al., 2019). We use a popular benchmark dataset called Penn Treebank (PTB) (Marcus et al., 1993) for training the LSTM network (**Figure 4D**) using MCA. The network consists of two LSTM modules, stacked with a final fully-connected layer, and has a total of 3.3 million trainable parameters (see **Supplementary Note 7**). The network is trained using sequences of text from the PTB dataset to predict the next character in the sequence. The network response is a probability distribution for the next character in the sequence. The performance in a character level language modeling task is commonly measured using bits-per-character (BPC), which is a measure of how well the model is able to predict samples from the true underlying probability distribution of the dataset. A lower BPC corresponds to a better model. The MCA based training and validation curves are shown in **Figure 4E** (see section A.3.4 for details). While the validation BPC from MCA and FP32 at the end of training are comparable, the difference between training and validation BPC is significantly smaller in MCA. The BPC obtained on the test set after MCA and FP32 training for different dropout rates are shown in **Figure 4F**. The optimal dropout rate that gives the lowest BPC for MCA is found to be lower than that for FP32, indicating regularization effects similar to those observed in the case of the CNN.

GANs are neural networks trained using an recently proposed adversarial training method (Goodfellow et al., 2014). The investigated network has two parts: a generator network that receives random noise as input and attempts to replicate the distribution of the training dataset and a discriminator network that attempts to distinguish between the training (real) images and the generated (fake) images. The network is deemed converged when the discriminator is no longer able to distinguish between the real and the fake images. Using the MNIST dataset, we successfully trained the GAN network shown in **Figure 4G** with MCA (see section A.3.6 for details). The performance of the generator to replicate the training dataset distribution is often evaluated using the Frechet distance (FD) (Liu et al., 2018). Even though the FD achieved by MCA training is slightly higher than that of FP32 training, the resulting generated images appear quite similar (see **Figure 4H**). The training of GANs is particularly sensitive to the mini-batch size and the choice of optimizers, even when training in FP32. As shown in **Figure 4I**, the solution converges to an optimal value only in the case of a mini-batch size of 100 and when stochastic gradient descent with momentum is used; we observed that the solution diverges in the other cases. Compared to alternate in-memory computing approaches where both the propagation and weight updates are performed in the analog domain (Ambrogio et al., 2018), a significant advantage of the proposed MCA approach is its ability to seamlessly incorporate these more sophisticated optimizers as well as the use of mini-batch sizes larger than one during training.

3. DISCUSSION

We demonstrated via experiments and simulations that the MCA can train PCM based analog synapses in DNNs to achieve accuracies comparable to those from the floating-point software training baselines. Here, we assess the overall system efficiency of the MCA for an exemplary problem and discuss pathways for achieving performance superiority as a general deep learning accelerator. We designed an application specific integrated circuit (ASIC) in 14 nm low power plus (14LPP) technology to perform the digital computations in the MCA and estimated the training energy per image, including that spent in the computational memory and the associated peripheral circuits (designed in 14LPP as well). The implementation was designed for the two-layer perceptron performing MNIST handwritten digit classification used in the experiments of section 2.3. For reference, an equivalent high-precision ASIC training engine was designed in 14LPP using 32-bit fixed-point format for data and computations with an effective throughput of 43k images/s at 0.62 W power consumption (see section A.4 and **Supplementary Note 8** for details). In both designs, all the digital memory necessary for training was implemented with on-chip static random-access memory. The MCA design resulted in 269× improvement in energy consumption for the forward and backward stages of training. Since the high-precision weight update computation and accumulation are the primary bottleneck for computational efficiency in the MCA, we implemented the outer-products for weight update computation using low-precision versions of the neuron activations and back-propagated errors (Hubara et al., 2017; Zhang et al., 2017; Wu et al., 2018), achieving comparable test accuracy with respect to the experiment of section 2.3 (see **Supplementary Note 8**). Activation and error vectors were represented using signed 3-bit numbers and shared scaling factors. The resulting weight update matrices were sparse with <1% non-zero entries on average. This proportionally reduced accesses to the 32-bit χ memory allocated for accumulating the weight updates. Necessary scaling operations for the non-zero entries were implemented using bit-shifts (Lin et al., 2016; Wu et al., 2018), thereby reducing the computing time and hardware complexity. Additional device programming overhead was negligible, since on average only one PCM device out of the array was programmed every two training images. This allowed the device programming to be executed in parallel to the weight update computation in the digital unit, without incurring additional time overhead. In contrast to the hardware experiment, the weight refresh operation was distributed across training examples as opposed to periodically refreshing the whole array during training, which allowed it to be performed in parallel with the weight update computation in the digital unit (see **Supplementary Note 8**). These optimizations resulted in 139× improvement in the energy consumption of the

TABLE 1 | Energy and time estimated based on application specific integrated circuit (ASIC) designs for processing one training image in MCA and corresponding fully digital 32-bit and mixed-precision designs.

Architecture	Parameter	Forward propagation	Backward propagation	Weight update	Total
32-bit design	Energy	5.62 μJ	0.09 μJ	8.64 μJ	14.35 μJ
	Time	7.31 μs	0.59 μs	15.36 μs	23.27 μs
Fully digital mixed-precision design	Energy	1.78 μJ	0.016 μJ	0.076 μJ	1.87 μJ
(4-bit weights, 8-bit activations/errors)	Time	6.41 μs	0.13 μs	0.79 μs	7.33 μs
MCA—computational memory	Energy	7.29 nJ	2.15 nJ	0.05 nJ	
	Time	0.27 μs	0.13 μs	–	
MCA—digital unit	Energy	8.97 nJ	2.76 nJ	61.98 nJ	
	Time	0.34 μs	0.09 μs	1.19 μs	
MCA—total	Energy	16.3 nJ	4.91 nJ	62.03 nJ	83.2 nJ
	Time	0.61 μs	0.22 μs	1.19 μs	2.02 μs

The numbers are for a specific two-layer perceptron with 785 input neurons, 250 hidden neurons, and 10 output neurons.

weight update stage in MCA with respect to the 32-bit design. Combining the three stages, the MCA consumed 83.2 nJ per image at 495 k images/s, resulting in a 11.5× higher throughput at 172× lower energy consumption with respect to the 32-bit implementation (see **Table 1**). We also compared the MCA with a fully digital mixed-precision ASIC in 14LPP, which uses 4-bit weights and 8-bit activations/errors for the data propagation stages. This design uses the same weight update implementation as the MCA design, but replaces the computational memory by a digital multiply-accumulate unit. The MCA achieves an overall energy efficiency gain of 22× with respect to this digital mixed-precision design (see section A.4). The energy estimates of the MCA on MNIST for inference-only (16.3 nJ/image) and training (83.2 nJ/image) may be on par or better compared with alternate architectures achieving similar accuracies (Bavandpour et al., 2018; Marinella et al., 2018; Chang et al., 2019; Park et al., 2019; Hirtzlin et al., 2020). For training, our approach compares favorably with a digital spiking neural network (254.3 nJ/image) (Park et al., 2019) and is also close to an analog-only approach using 3T1C+PCM synaptic devices (48 nJ/image) (Chang et al., 2019). However, there are notable differences in the technology nodes, methods of energy estimation, actual computations and network size involved in each design, making accurate comparisons with those alternate approaches difficult to provide.

While the above study was limited to a simple two-layer perceptron, deep learning with MCA could generally have the following benefits over fully digital implementations. For larger networks, digital deep learning accelerators as well as the MCA will have to rely on dynamic random-access memory (DRAM) to store the model parameters, activations, and errors, which will significantly increase the cost of access to those variables compared with on-chip SRAM. Hence, implementing DNN weights in nanoscale memory devices could enable large neural networks to be fit on-chip without expensive off-chip communication during the data propagations. Analog crossbar arrays implementing matrix-vector multiplications in $\mathcal{O}(1)$ time permit orders of magnitude computational acceleration of the data-propagation stages (Gokmen and Vlasov, 2016; Li et al., 2018b; Merrikh-Bayat et al., 2018). Analog in-memory processing is a desirable trade-off of numerical precision for computational energy efficiency, as a growing number of DNN architectures are being demonstrated to support low precision weights (Courbariaux et al., 2015; Choi et al., 2019). In contrast to digital mixed-precision ASIC implementations, where the same resources are shared among all the computations, the dedicated weight layers in MCA permit more efficient inter and intra layer pipelines (Shafiee et al., 2016; Song et al., 2017). Handling the control of such pipelines for training various network topologies adequately with optimized array-to-array communication, which is a non-trivial task, will be crucial in harnessing the efficiency of the MCA for deep learning. Compared with the fully analog accelerators being explored (Agarwal et al., 2017; Kim et al., 2017; Ambrogio et al., 2018), the MCA requires an additional high-precision digital memory of same size as the model, and the need to access that memory during the weight update stage. However, the digital implementation of the optimizer in the MCA provides high-precision gradient accumulation and the flexibility to realize a wide class of optimization algorithms, which are highly desirable in a general deep learning accelerator. Moreover, the MCA significantly relaxes the analog synaptic device requirements, particularly those related to linearity, variability, and update precision to realize high-performance learning machines. In contrast to the periodic carry approach (Agarwal et al., 2017; Ambrogio et al., 2018), it avoids the need of reprogramming all the weights at specific intervals during training (except for a small number of devices during weight refresh). Instead, single-shot blind pulses are applied to chosen synapses at every weight update, resulting in sparse device programming. This relaxes the overall reliability and endurance requirements of the nanoscale devices and reduces the time and energy spent to program them.

In summary, we proposed a mixed-precision computational memory architecture for training DNNs and experimentally demonstrated its ability to deliver performance close to equivalent 64-bit floating-point simulations. We used a prototype phase-change memory (PCM) chip to perform the training of a two-layer perceptron containing 198,760 synapses on the MNIST dataset. We further validated the approach by training a CNN on the CIFAR-10 dataset, an LSTM network on the Penn Treebank dataset, and a GAN to generate MNIST digits. The training of these larger networks was performed through simulations using a PCM behavioral model that matches the characteristics of our prototype array, and achieved accuracy close to 32-bit software training in all the three cases. The proposed architecture decouples the optimization algorithm and its hyperparameters from the device update precision, allowing it to be employed with a wide range of non-volatile memory devices without resorting to complex device-specific hyperparameter tuning for achieving satisfactory learning performance. We also showed evidence for inherent regularization effects originating from the non-linear and stochastic behavior of these devices that is indicative of futuristic learning machines exploiting rather than overcoming the underlying operating characteristics of nanoscale devices. These results show that the proposed architecture can be used to train a wide range of DNNs in a reliable and flexible manner with existing memristive devices, offering a pathway toward more energy-efficient deep learning than with general-purpose computing systems.

DATA AVAILABILITY STATEMENT

The datasets generated for this study are available on request to the corresponding author.

AUTHOR CONTRIBUTIONS

SN, ML, IB, AS, and EE conceived the mixed-precision computational memory architecture for deep learning. SN developed the PCM model and performed the hardware experiments with the support of ML and IB. CP, VJ, and GM developed the TensorFlow simulator and performed the training simulations of the large networks. GK designed the 32-bit digital training ASIC and the digital unit of the MCA, and performed their energy estimations. RK-A designed the computational memory unit of the MCA and performed its energy estimation. UE, AP, and TA designed, built, and developed the software of the hardware platform hosting the prototype PCM chip used in the experiments. SN, ML, and AS wrote the manuscript with input from all authors. ML, BR, AS, and EE supervised the project.

ACKNOWLEDGMENTS

We thank C. Malossi, N. Papandreou, and C. Bekas for discussions. M. Braendli and M. Stanisavljevic for help with digital design tools, and our colleagues in IBM TJ Watson Research Center, in particular M. BrightSky, for help with fabricating the PCM prototype chip used in this work. AS acknowledges funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement number 682675).

SUPPLEMENTARY MATERIAL

The Supplementary Material for this article can be found online at: https://www.frontiersin.org/articles/10.3389/fnins.2020.00406/full#supplementary-material

REFERENCES

Agarwal, S., Gedrim, R. B. J., Hsia, A. H., Hughart, D. R., Fuller, E. J., Talin, A. A., et al. (2017). "Achieving ideal accuracies in analog neuromorphic computing using periodic carry," in *2017 Symposium on VLSI Technology* (Kyoto), T174–T175. doi: 10.23919/VLSIT.2017.7998164

Ambrogio, S., Narayanan, P., Tsai, H., Shelby, R. M., Boybat, I., Nolfo, C., et al. (2018). Equivalent-accuracy accelerated neural-network training using analogue memory. *Nature* 558:60. doi: 10.1038/s41586-018-0180-5

Arjovsky, M., Chintala, S., and Bottou, L. (2017). Wasserstein GAN. *arXiv [Preprint]. arXiv:1701.07875*.

Bavandpour, M., Mahmoodi, M. R., Nili, H., Bayat, F. M., Prezioso, M., Vincent, A., et al. (2018). "Mixed-signal neuromorphic inference accelerators: recent results and future prospects," in *IEEE International Electron Devices Meeting (IEDM)* (San Francisco, CA), 20.4.1–20.4.4. doi: 10.1109/IEDM.2018.8614659

Boybat, I., Le Gallo, M., Nandakumar, S., Moraitis, T., Parnell, T., Tuma, T., et al. (2018a). Neuromorphic computing with multi-memristive synapses. *Nat. Commun*. 9:2514. doi: 10.1038/s41467-018-04933-y

Boybat, I., Nandakumar, S. R., Gallo, M. L., Rajendran, B., Leblebici, Y., Sebastian, A., et al. (2018b). "Impact of conductance drift on multi-PCM synaptic architectures," in *2018 Non-Volatile Memory Technology Symposium (NVMTS)* (Sendai), 1–4. doi: 10.1109/NVMTS.2018.8603100

Breitwisch, M., Nirschl, T., Chen, C., Zhu, Y., Lee, M., Lamorey, M., et al. (2007). "Novel lithography-independent pore phase change memory," in *Proc. IEEE Symposium on VLSI Technology* (Kyoto), 100–101. doi: 10.1109/VLSIT.2007.4339743

Burr, G. W., Brightsky, M. J., Sebastian, A., Cheng, H.-Y., Wu, J.-Y., Kim, S., et al. (2016). Recent progress in phase-change memory technology. *IEEE J. Emerg. Select. Top. Circ. Syst*. 6, 146–162. doi: 10.1109/JETCAS.2016.2547718

Burr, G. W., Shelby, R. M., Sebastian, A., Kim, S., Kim, S., Sidler, S., et al. (2017). Neuromorphic computing using non-volatile memory. *Adv. Phys*. 2, 89–124. doi: 10.1080/23746149.2016.1259585

Burr, G. W., Shelby, R. M., Sidler, S., Di Nolfo, C., Jang, J., Boybat, I., et al. (2015). Experimental demonstration and tolerancing of a large-scale neural network (165 000 synapses) using phase-change memory as the synaptic weight element. *IEEE Trans. Electr. Dev*. 62, 3498–3507. doi: 10.1109/TED.2015.2439635

Cassinerio, M., Ciocchini, N., and Ielmini, D. (2013). Logic computation in phase change materials by threshold and memory switching. *Adv. Mater*. 25, 5975–5980. doi: 10.1002/adma.201301940

Chang, H. Y., Narayanan, P., Lewis, S. C., Farinha, N. C. P., Hosokawa, K., Mackin, C., et al. (2019). AI hardware acceleration with analog memory: Microarchitectures for low energy at high speed. *IBM J. Res. Dev*. 63, 8:1–8:14. doi: 10.1147/JRD.2019.2934050

Choi, J., Venkataramani, S., Srinivasan, V., Gopalakrishnan, K., Wang, Z., and Chuang, P. (2019). Accurate and efficient 2-bit quantized neural networks. In *Proceedings of the 2nd SysML Conference* (Palo Alto, CA). Available online at: https://mlsys.org/Conferences/2019/doc/2019/168.pdf

Close, G., Frey, U., Breitwisch, M., Lung, H., Lam, C., Hagleitner, C., et al. (2010). "Device, circuit and system-level analysis of noise in multi-bit phase-change memory," in *2010 IEEE International Electron Devices Meeting (IEDM)* (San Francisco, CA). doi: 10.1109/IEDM.2010.5703445

Courbariaux, M., Bengio, Y., and David, J.-P. (2015). "Binaryconnect: Training deep neural networks with binary weights during propagations," in *Advances in Neural Information Processing Systems* (Montreal, BC), 3123–3131.

Glorot, X., and Bengio, Y. (2010). "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, Vol. 9 of Proceedings of Machine Learning Research*, eds Y. W. Teh and M. Titterington (Sardinia), 249–256.

Gokmen, T., Onen, M., and Haensch, W. (2017). Training deep convolutional neural networks with resistive cross-point devices. *Front. Neurosci.* 11:538. doi: 10.3389/fnins.2017.00538

Gokmen, T., Rasch, M., and Haensch, W. (2018). Training LSTM networks with resistive cross-point devices. *Front. Neurosci.* 12:745. doi: 10.3389/fnins.2018.00745

Gokmen, T., and Vlasov, Y. (2016). Acceleration of deep neural network training with resistive cross-point devices: design considerations. *Front. Neurosci.* 10:333. doi: 10.3389/fnins.2016.00333

Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., et al. (2014). "Generative adversarial nets," in *Proceedings of the 27th International Conference on Neural Information Processing Systems - Vol. 2, NIPS'14* (Cambridge, MA: MIT Press), 2672–2680.

Gupta, S., Agrawal, A., Gopalakrishnan, K., and Narayanan, P. (2015). "Deep learning with limited numerical precision," in *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)* (Lille), 1737–1746.

Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., and Hochreiter, S. (2017). "GANs trained by a two time-scale update rule converge to a local nash equilibrium," in *Advances in Neural Information Processing Systems* (Long Beach, CA), 6626–6637.

Hirtzlin, T., Bocquet, M., Penkovsky, B., Klein, J.-O., Nowak, E., Vianello, E., et al. (2020). Digital biologically plausible implementation of binarized neural networks with differential hafnium oxide resistive memory arrays. *Front. Neurosci.* 13:1383. doi: 10.3389/fnins.2019.01383

Hu, M., Strachan, J. P., Li, Z., Grafals, E. M., Davila, N., Graves, C., et al. (2016). "Dot-product engine for neuromorphic computing: programming 1T1M crossbar to accelerate matrix-vector multiplication," in *53nd ACM/EDAC/IEEE Design Automation Conference (DAC)* (Austin, TX), 1–6. doi: 10.1145/2897937.2898010

Hubara, I., Courbariaux, M., Soudry, D., El-Yaniv, R., and Bengio, Y. (2017). Quantized neural networks: training neural networks with low precision weights and activations. *J. Mach. Learn. Res.* 18, 6869–6898. doi: 10.5555/3122009.3242044

Ielmini, D., and Wong, H.-S. P. (2018). In-memory computing with resistive switching devices. *Nat. Electron.* 1:333. doi: 10.1038/s41928-018-0092-2

Ioffe, S., and Szegedy, C. (2015). "Batch normalization: accelerating deep network training by reducing internal covariate shift," in *Proceedings of the 32nd International Conference on Machine Learning, Vol. 37* (Lille: JMLR.org), 448–456. doi: 10.5555/3045118.3045167

Joshi, V., Le Gallo, M., Haefeli, S., Boybat, I., Nandakumar, S. R., Piveteau, C. et al. (2020). Accurate deep neural network inference using computational phase-change memory. *Nat. Commun.* in press. *arXiv: 1906.03138.*

Kaur, P. (2017). *Convolutional Neural Networks (CNN) for CIFAR-10 Dataset*. Available online at: http://parneetk.github.io/blog/cnn-cifar10/

Kim, S., Gokmen, T., Lee, H., and Haensch, W. E. (2017). "Analog CMOS-based resistive processing unit for deep neural network training," in *2017 IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS)* (Boston, MA), 422–425. doi: 10.1109/MWSCAS.2017.8052950

Kingma, D. P., Ba, J. (2015). "Adam: a method for stochastic optimization," in *3rd International Conference on Learning Representations, (ICLR) 2015*, eds Y. Bengio and Y. LeCun (San Diego, CA). Available online at: http://arxiv.org/abs/1412.6980

Kuzum, D., Jeyasingh, R. G., Lee, B., and Wong, H.-S. P. (2011). Nanoelectronic programmable synapses based on phase change materials for brain-inspired computing. *Nano Lett.* 12, 2179–2186. doi: 10.1021/nl201040y

Le Gallo, M., Krebs, D., Zipoli, F., Salinga, M., and Sebastian, A. (2018). Collective structural relaxation in phase-change memory devices. *Adv. Electr. Mater.* 4:1700627. doi: 10.1002/aelm.201700627

Le Gallo, M., Sebastian, A., Cherubini, G., Giefers, H., and Eleftheriou, E. (2018a). Compressed sensing with approximate message passing using in-memory computing. *IEEE Trans. Electr. Dev.* 65, 4304–4312. doi: 10.1109/TED.2018.2865352

Le Gallo, M., Sebastian, A., Mathis, R., Manica, M., Giefers, H., Tuma, T., et al. (2018b). Mixed-precision in-memory computing. *Nat. Electron.* 1:246. doi: 10.1038/s41928-018-0054-8

Le Gallo, M., Tuma, T., Zipoli, F., Sebastian, A., and Eleftheriou, E. (2016). "Inherent stochasticity in phase-change memory devices," in *46th European Solid-State Device Research Conference (ESSDERC)* (Lausanne), 373–376. doi: 10.1109/ESSDERC.2016.7599664

LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature* 521, 436–444. doi: 10.1038/nature14539

Li, C., Belkin, D., Li, Y., Yan, P., Hu, M., Ge, N., et al. (2018a). Efficient and self-adaptive in-situ learning in multilayer memristor neural networks. *Nat. Commun.* 9:2385. doi: 10.1038/s41467-018-04484-2

Li, C., Hu, M., Li, Y., Jiang, H., Ge, N., Montgomery, E., et al. (2018b). Analogue signal and image processing with large memristor crossbars. *Nat. Electron.* 1:52. doi: 10.1038/s41928-017-0002-z

Li, C., Wang, Z., Rao, M., Belkin, D., Song, W., Jiang, H., et al. (2019). Long short-term memory networks in memristor crossbar arrays. *Nat. Mach. Intell.* 1:49. doi: 10.1038/s42256-018-0001-4

Lin, Z., Courbariaux, M., Memisevic, R., and Bengio, Y. (2016). "Neural Networks with Few Multiplications," in *4th International Conference on Learning Representations, (ICLR) 2016*, eds Y. Bengio and Y. LeCun (San Juan). Available online at: http://arxiv.org/abs/1510.03009

Liu, S., Wei, Y., Lu, J., and Zhou, J. (2018a). An improved evaluation framework for generative adversarial networks. *arXiv [Preprint]. arXiv:abs/1803.07474*.

Lucic, M., Kurach, K., Michalski, M., Bousquet, O., and Gelly, S. (2018). "Are GANs created equal? A large-scale study," in *Proceedings of the 32nd International Conference on Neural Information Processing Systems* (Red Hook, NY: Curran Associates Inc.), 698–707. doi: 10.5555/3326943.3327008

Marcus, M. P., Marcinkiewicz, M. A., and Santorini, B. (1993). Building a large annotated corpus of English: the Penn Treebank. *Comput. Linguist.* 19, 313–330. doi: 10.21236/ADA273556

Marinella, M. J., Agarwal, S., Hsia, A., Richter, I., Jacobs-Gedrim, R., Niroula, J., et al. (2018). Multiscale co-design analysis of energy, latency, area, and accuracy of a ReRAM analog neural training accelerator. *IEEE J. Emerg. Select. Top. Circ. Syst.* 8, 86–101. doi: 10.1109/JETCAS.2018.2796379

Merity, S., Keskar, N. S., and Socher, R. (2018). An analysis of neural language modeling at multiple scales. *arXiv [Preprint]. arXiv:1803.08240.*

Merolla, P., Appuswamy, R., Arthur, J., Esser, S. K., and Modha, D. (2016). Deep neural networks are robust to weight binarization and other non-linear distortions. *arXiv [Preprint]. arXiv:1606.01981.*

Merrikh-Bayat, F., Guo, X., Klachko, M., Prezioso, M., Likharev, K. K., and Strukov, D. B. (2018). High-performance mixed-signal neurocomputing with nanoscale floating-gate memory cell arrays. *IEEE Trans. Neural Netw. Learn. Syst.* 29, 4782–4790. doi: 10.1109/TNNLS.2017.2778940

Micikevicius, P., Narang, S., Alben, J., Diamos, G. F., Elsen, E., García, D., et al. (2018). "Mixed precision training," in *6th International Conference on Learning Representations (ICLR)* (Vancouver, BC: OpenReview.net). Available online at: https://openreview.net/forum?id=r1gs9JgRZ

Nandakumar, S., Kulkarni, S. R., Babu, A. V., and Rajendran, B. (2018a). Building brain-inspired computing systems: examining the role of nanoscale devices. *IEEE Nanotechnol. Mag.* 12, 19–35. doi: 10.1109/MNANO.2018.2845078

Nandakumar, S., Le Gallo, M., Boybat, I., Rajendran, B., Sebastian, A., and Eleftheriou, E. (2018b). "Mixed-precision architecture based on computational memory for training deep neural networks," in *International Symposium on Circuits and Systems (ISCAS)* (Florence), 1–5. doi: 10.1109/ISCAS.2018.8351656

Nandakumar, S., Le Gallo, M., Boybat, I., Rajendran, B., Sebastian, A., and Eleftheriou, E. (2018c). A phase-change memory model for neuromorphic computing. *J. Appl. Phys.* 124:152135. doi: 10.1063/1.5042408

Nardone, M., Kozub, V., Karpov, I., and Karpov, V. (2009). Possible mechanisms for 1/f noise in chalcogenide glasses: a theoretical description. *Phys. Rev. B* 79:165206. doi: 10.1103/PhysRevB.79.165206

Papandreou, N., Pozidis, H., Pantazi, A., Sebastian, A., Breitwisch, M., Lam, C., et al. (2011). "Programming algorithms for multilevel phase-change memory," in *IEEE International Symposium on Circuits and Systems (ISCAS)* (Rio de Janeiro), 329–332. doi: 10.1109/ISCAS.2011.5937569

Park, J., Lee, J., and Jeon, D. (2019). "7.6 A 65nm 236.5nJ/classification neuromorphic processor with 7.5% energy overhead on-chip learning using direct spike-only feedback," in *2019 IEEE International Solid-State Circuits Conference - (ISSCC)* (San Francisco, CA), 140–142. doi: 10.1109/ISSCC.2019.8662398

Prezioso, M., Merrikh-Bayat, F., Hoskins, B., Adam, G. C., Likharev, K. K., and Strukov, D. B. (2015). Training and operation of an integrated neuromorphic network based on metal-oxide memristors. *Nature* 521:61. doi: 10.1038/nature14441

Sebastian, A., Le Gallo, M., Burr, G. W., Kim, S., BrightSky, M., and Eleftheriou, E. (2018). Tutorial: brain-inspired computing using phase-change memory devices. *J. Appl. Phys*. 124:111101. doi: 10.1063/1.5042413

Sebastian, A., Le Gallo, M., Khaddam-Aljameh, R., Eleftheriou, E. (2020). Memory devices and applications for in-memory computing. *Nat. Nanotechnol.* doi: 10.1038/s41565-020-0655-z

Sebastian, A., Le Gallo, M., and Krebs, D. (2014). Crystal growth within a phase change memory cell. *Nat. Commun*. 5:4314. doi: 10.1038/ncomms5314

Sebastian, A., Tuma, T., Papandreou, N., Le Gallo, M., Kull, L., Parnell, T., et al. (2017). Temporal correlation detection using computational phase-change memory. *Nat. Commun*. 8:1115. doi: 10.1038/s41467-017-01481-9

Seshadri, V., Lee, D., Mullins, T., Hassan, H., Boroumand, A., Kim, J., et al. (2017). "Ambit: in-memory accelerator for bulk bitwise operations using commodity DRAM technology," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture* (New York, NY: Association for Computing Machinery), 273–287. doi: 10.1145/3123939.3124544

Shafiee, A., Nag, A., Muralimanohar, N., Balasubramonian, R., Strachan, J. P., Hu, M., et al. (2016). "ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)* (Seoul), 14–26. doi: 10.1109/ISCA.2016.12

Sheridan, P. M., Cai, F., Du, C., Ma, W., Zhang, Z., and Lu, W. D. (2017). Sparse coding with memristor networks. *Nat. Nanotechnol*. 12:784. doi: 10.1038/nnano.2017.83

Shor, J. (2017). *TFGAN*. Available online at: https://github.com/tensorflow/models/tree/master/research/gan/mnist/data

Song, L., Qian, X., Li, H., and Chen, Y. (2017). "Pipelayer: A pipelined ReRAM-based accelerator for deep learning," in *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)* (Austin, TX), 541–552. doi: 10.1109/HPCA.2017.55

Sun, X., Wang, P., Ni, K., Datta, S., and Yu, S. (2018). "Exploiting hybrid precision for training and inference: a 2T-1FeFET based analog synaptic weight cell," in *2018 IEEE International Electron Devices Meeting (IEDM)* (San Francisco, CA), 3.1.1–3.1.4. doi: 10.1109/IEDM.2018.8614611

Tang, Y. (2013). Deep learning using linear support vector machines. *arXiv [Preprint]. arXiv:1306.0239*.

Tuma, T., Pantazi, A., Le Gallo, M., Sebastian, A., and Eleftheriou, E. (2016). Stochastic phase-change neurons. *Nat. Nanotechnol*. 11, 693–699. doi: 10.1038/nnano.2016.70

Wong, H. S. P., and Salahuddin, S. (2015). Memory leads the way to better computing. *Nat. Nanotechnol*. 10, 191–194. doi: 10.1038/nnano.2015.29

Wouters, D. J., Waser, R., and Wuttig, M. (2015). Phase-change and redox-based resistive switching memories. *Proc. IEEE* 103, 1274–1288. doi: 10.1109/JPROC.2015.2433311

Wu, S., Li, G., Feng, C., Shi, L. (2018). "Training and inference with integers in deep neural networks," in *6th International Conference on Learning Representations, ICLR 2018* (Vancouver, BC: OpenReview.net). Available online at: https://openreview.net/forum?id=HJGXzmspb

Yao, P., Wu, H., Gao, B., Eryilmaz, S. B., Huang, X., Zhang, W., et al. (2017). Face classification using electronic synapses. *Nat. Commun*. 8:15199. doi: 10.1038/ncomms15199

Yu, S. (2018). Neuro-inspired computing with emerging nonvolatile memory. *Proc. IEEE* 106, 260–285. doi: 10.1109/JPROC.2018.2790840

Zaremba, W., Sutskever, I., and Vinyals, O. (2014). Recurrent neural network regularization. *arXiv [Preprint]. arXiv:1409.2329*.

Zhang, H., Li, J., Kara, K., Alistarh, D., Liu, J., and Zhang, C. (2017). "ZipML: Training linear models with end-to-end low precision, and a little bit of deep learning," in *Proceedings of the 34th International Conference on Machine Learning (ICML), Vol. 70* (Sydney, NSW), 4035–4043.

Conflict of Interest: SN, ML, CP, VJ, GM, IB, GK, RK-A, UE, AP, AS, and EE were employed by the company IBM Research-Zurich.

The remaining authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

A. METHODS

A.1. PCM-Based Hardware Platform

The experimental hardware platform is built around a prototype phase-change memory (PCM) chip that contains 3 million PCM devices (Close et al., 2010). The PCM devices are based on doped $Ge_2Sb_2Te_5$ (GST) and are integrated into the chip in 90 nm CMOS baseline technology. In addition to the PCM devices, the chip integrates the circuitry for device addressing, on-chip ADC for device readout, and voltage- or current-mode device programming. The experimental platform comprises a high-performance analog-front-end (AFE) board that contains a number of digital-to-analog converters (DACs) along with discrete electronics such as power supplies, voltage and current reference sources. It also comprises an FPGA board that implements the data acquisition and the digital logic to interface with the PCM device under test and with all the electronics of the AFE board. The FPGA board also contains an embedded processor and ethernet connection that implement the overall system control and data management as well as the interface with the host computer. The embedded microcode allows the execution of the multi-device programming and readout experiments that implement the matrix-vector multiplications and weight updates on the PCM chip. The hardware modules implement the interface with the external DACs used to provide various voltages to the chip for programming and readout, as well as the interface with the memory device under test, i.e., the addressing interface, the programming-mode, or read-mode interfaces, etc.

The PCM device array is organized as a matrix of 512 word lines (WL) and 2,048 bit lines (BL). Each individual device along with its access transistor occupies an area of $50\,F^2$ (F is the technology feature size, $F = 90\,\text{nm}$). The PCM devices were integrated into the chip in 90 nm CMOS technology using a sub-lithographic key-hole transfer process (Breitwisch et al., 2007). The bottom electrode has a radius of $\sim$ 20 nm and a length of $\sim$ 65 nm. The phase change material is $\sim$ 100 nm thick and extends to the top electrode, whose radius is $\sim$ 100 nm. The selection of one PCM device is done by serially addressing a WL and a BL. The addresses are decoded and they then drive the WL driver and the BL multiplexer. The single selected device can be programmed by forcing a current through the BL with a voltage-controlled current source. For reading a PCM device, the selected BL is biased to a constant voltage of 300 mV by a voltage regulator via a voltage V_{read} generated off-chip. The sensed current, I_{read}, is integrated by a capacitor, and the resulting voltage is then digitized by the on-chip 8-bit cyclic ADC. The total time of one read is $1\,\mu$s. The readout characteristic is calibrated via the use of on-chip reference polysilicon resistors. For programming a PCM device, a voltage V_{prog} generated off-chip is converted on-chip into a programming current, I_{prog}. This current is then mirrored into the selected BL for the desired duration of the programming pulse. Each programming pulse is a box-type rectangular pulse with duration of 10 ns to 400 ns and amplitude varying between 0 and $500\,\mu$A. The access-device gate voltage (WL voltage) is kept high at 2.75 V during programming. Iterative programming, which is used for device initialization in our experiments, is achieved by applying a sequence of programming pulses (Papandreou et al., 2011). After each programming pulse, a verify step is performed and the value of the device conductance programmed in the preceding iteration is read at a voltage of 0.2 V. The programming current applied to the PCM device in the subsequent iteration is adapted according to the sign of the value of the error between the target level and read value of the device conductance. The programming sequence ends when the error between the target conductance and the programmed conductance of the device is smaller than a desired margin or when the maximum number of iterations (20) has been reached. The total time of one program-and-verify step is approximately $2.5\,\mu$s.

A.2. Mixed-Precision Training Experiment

A.2.1. Network Details

The network used in the experiment had 784 inputs and 1 bias at the input layer, 250 sigmoid neurons and 1 bias at the hidden layer, and 10 sigmoid neurons at the output layer. The network was trained by minimizing the mean square error loss function with stochastic gradient descent (SGD). The network was trained with a batch size of 1, meaning that the weight updates were computed after every training example. We used a fixed learning rate of 0.4. We used the full MNIST training dataset of 60,000 images for training the network, and the test dataset of 10,000 images for computing the test accuracy. The order of the images was randomized for each training epoch. Apart from normalizing the gray-scale images, no additional pre-processing was performed on the training and test sets.

A.2.2. Differential PCM Experiment

397,520 devices were used from the PCM hardware platform to represent the two-layer network (**Figure 3A**) weights in a differential configuration. The devices were initialized to a conductance distribution with mean of $1.6\,\mu$S and standard deviation of $0.83\,\mu$S via iterative programming. Since the platform allowed only serial access to the devices, all the conductance values were read from hardware and reported to the software that performed the forward and backward propagations. The weight updates were computed and accumulated in the χ memory in software. When the magnitude of χ exceeded ϵ for a particular weight, a 50 ns pulse with an amplitude of $90\,\mu$A was applied to the corresponding device (G_p or G_n, depending on the sign of χ) of the PCM chip. The synaptic conductance to weight conversion was performed by a linear mapping between $[-8\,\mu\text{S}, 8\,\mu\text{S}]$ in the conductance domain and $[-1, 1]$ in the weight domain.

The PCM devices exhibit temporal variations in the conductance values such as conductance drift and read noise. As a result, each matrix multiplication in every layer will see a slightly different weight matrix even in the absence of any weight update. However, the cost of re-reading the entire conductance array for every matrix-vector multiplication in our experiment was prohibitively large due to the serial interface. Therefore, after each programming event to the PCM array, we read the conductance values of a subset of all the PCM devices along with the programmed device. Specifically, we read all the devices in the second layer and a set of 785 pairs of devices from the first layer in a round robin fashion after every device programming event. This approach faithfully captures the effects of PCM hardware noise and drift in the network propagations during training. For the weight refresh operation, the conductance pairs in software were verified every 100 training examples and if one of the device conductance values was above 8 μS and if their difference was less than 6 μS, both the devices were RESET using 500 ns, 360 μA pulses and their difference was converted to a number of SET pulses based on an average observed conductance change per pulse. During this weight refresh, the maximum number of pulses was limited to 3 and the pulses were applied to G_p or G_n depending on the sign of their initial conductance difference.

A.2.3. Inference After Training

After training, all the PCM conductance values realizing the weights of the two-layer perceptron are read at different time intervals and used to evaluate the classification accuracy on the 60,000 MNIST training images and 10,000 test images. Despite the conductance drift, there was only negligible accuracy drop over a month. The following factors might be contributing to this drift tolerance. During on-chip training, different devices are programmed at different times and are drifting during the training process. The training algorithm could compensate for the error created by the conductance drift and could eventually generate a more drift resilient solution. Furthermore, the perceptron weights are implemented using the conductance difference of two PCM devices. This differential configuration partially compensates the effect of drift (Boybat et al., 2018b). Also, from the empirical relation for the conductance drift, we have $\frac{dG}{dt} = G(t)\frac{(-\nu)}{t} \propto \frac{1}{t}$, which means that the conductance decay over time decreases as we advance in time. Drift compensation strategies such as using a global scaling factor (Le Gallo et al., 2018a; Joshi et al., 2020) could also be used in more complex deep learning models to maintain the accuracy over extended periods of time.

A.2.4. Non-differential PCM Experiment

A non-differential PCM configuration for the synapse was tested in the mixed-precision training architecture to implement both weight increment and decrement by programming the same device in either direction and hence avoid the conductance saturation and the associated weight refresh overhead. The non-accumulative RESET behavior of the PCM device makes its conductance potentiation and depression highly asymmetric and hence this experiment also validates tolerance of the mixed-precision training architecture to such programming asymmetry. We conducted the same training experiment as before for the MNIST digit classification, except that now each synapse was realized using a single PCM with a reference level to realize bipolar weights. The experiment requires 198,760 PCM devices. Potentiation is implemented by 90 μA, 50 ns SET pulses and depression is implemented using 400 μA, 50 ns RESET pulses. In the mixed-precision architecture, this asymmetric conductance update behavior is compensated for by using different ϵs for potentiation and depression. We used ϵ_P corresponding to 0.77 μS for weight increment and ϵ_D corresponding to 8 μS for weight decrement.

In order to achieve bipolar weights with non-differential synapses, a reference conductance level must be introduced. Ideally, this reference level could be implemented using any resistive device which is one time programmed to the necessary conductance level. In the case of PCM devices, due to their conductance drift, it is more suitable to implement the reference level using the PCM technology which follows the same average drift behavior of the devices that represent the synapses. In this experiment, we used the average conductance of all the PCM devices read from the array to represent the reference conductance level (G_{ref}), and hence the network weights $W \propto (G - G_{ref})$. This reference level may be potentially realized in a crossbar architecture by reserving a column on which all the devices are programmed to G_{ref}. G_{ref} corresponds to half the device conductance range and denotes the conductance that will be mapped to zero weight value. The current from the G_{ref} column can be subtracted from the currents of the columns of synaptic devices. For the experiment, the devices are initialized to a distribution with mean conductance of 4.5 μS and standard deviation of 1.25 μS. While a narrower distribution was desirable (Glorot and Bengio, 2010), it was difficult to achieve in the chosen initialization range. However, this was compensated by mapping the conductance values to a narrower weight range at the beginning of the training. This weight range is progressively relaxed over the next few epochs. The conductance range [0.1, 8] μS is mapped to [−0.7, 0.7] during epoch 1. Thereafter, the range is incremented in uniform steps/epoch to [−1, 1] by epoch 3 and is held constant thereafter. Also, the mean value of the initial conductance distribution was chosen to be slightly higher than the midpoint of the conductance range to compensate for the conductance drift. Irrespective of the conductance change asymmetry, the training in MCA achieves a maximum training accuracy of 98.77% and a maximum test accuracy of 97.47% in 30 epochs (see **Supplementary Figure 1**).

A.3. Training Simulations of Larger Networks With PCM Model

A.3.1. Simulator

The simulator is implemented as an extension to the Tensorflow deep learning framework. The Tensorflow operations that can be implemented on computational memory are replaced with custom implementations. For example, the Ohm's law and Kirchhoff's circuit laws replace the matrix-vector multiplications and are implemented based on the PCM model described in section 2.2. The various non-idealities associated with the PCM devices such as limited conductance range, read noise and conductance drift as well as the quantization effects arising from the data converters (with 8-bit quantization) are incorporated while performing the matrix-vector multiplications. The synaptic weight update routines are also replaced with custom ones that implement the mixed-precision weight update accumulation and stochastic conductance update using the model of the PCM described in section 2.2. In the matrix-vector multiplication operations, the weight matrix represented using stochastic PCM devices is multiplied by 8-bit fixed-point neuron activations or normalized 8-bit fixed-point error vectors. Since the analog current accumulation along the wires in the computational memory unit can be assumed to have arbitrary precision, the matrix-vector multiplication between the noisy modeled PCM weights and quantized inputs is computed in 32-bit floating-point. In order to model the peripheral analog to digital conversion, the matrix-vector multiplication results are quantized back to 8-bit fixed-point. We evaluated the effect of ADC precision on DNN training performance based on the MNIST handwritten digit classification problem (Nandakumar et al., 2018b). The accuracy loss due to quantization was estimated to be approximately 0.1% in MCA for 8-bit ADC, which progressively increases with reduced precision. It is possible to reduce the ADC precision down to 4-bit at the cost of a few percentage drop in accuracy for reduced circuit complexity. However, we chose to maintain an 8-bit ADC precision for the training of networks to maintain comparable accuracies with the 32-bit precision baseline. The remaining training operations such as activations, dropout, pooling, and weight update computations use 32-bit floating-point precision.

Hence, these operations use original Tensorflow implementations in the current simulations and can be performed in the digital unit of the MCA in its eventual hardware implementation. Additional details on the simulator can be found in **Supplementary Note 6**.

A.3.2. CNN

The CNN used for the simulation study has nine layers—six convolution layers and three fully-connected layers (Kaur, 2017). A pooling layer is inserted after every two convolution layers. Together, the network has approximately 1.5 million parameters. ReLU activation is used at all convolution and fully-connected layers and softmax activation is used at the output layer. Dropout regularization technique was employed after the pooling layers and in-between the fully-connected layers (see **Supplementary Note 7**). To map the convolution kernels to crossbar arrays, the filters are stretched out to 1D arrays and horizontally stacked on the memristive crossbar (Gokmen et al., 2017). The different image patches are extracted from the input image, stretched out and finally rearranged to form the columns of a large matrix. The convolution can now be computed by performing the matrix-matrix multiplication between these two matrices and subsequently, reordering the output. In other words, the forward propagation of neuron activations and the backward propagation of the errors can be implemented as matrix-vector multiplications. During the extraction process of the image patches, the original image is padded with zero pixels at the border, to ensure that the output of the convolution layer has the same image size as the input. Considering an input image of size $n \times n$ and d channels and m convolution kernels of size $k \times k$, the dimensions of the input matrix is $dk^2 \times n^2$ and the dimensions of the matrix on the crossbar array is $dk^2 \times m$. The convolution operation can therefore be performed in n^2 matrix-vector multiplication cycles.

The FP32 baseline and the MCA implementation of the CNN were trained using SGD with a minibatch size of 100 and cross-entropy loss function. Light data augmentation was employed in the form of random image flipping and random adjustments of brightness and contrast to the training dataset. We used a learning rate of 0.03, which was dropped by a factor of 0.1 at epoch 200. In the MCA implementation, the weight updates computed from all the weight layers were accumulated in χ and were subsequently transferred to the modeled PCM devices organized in crossbar arrays (see **Supplementary Note 7**). The pooling operations of the CNN were performed in the conventional digital domain. Conductance refresh was performed after every 51 batches in the MCA implementation.

A.3.3. Testing of the Regularization Effect Observed During Training of the CNN

We observed that MCA achieves higher test accuracy with lower training accuracy compared to the FP32 training. This is a desirable effect referred to as regularization and techniques such as dropout are typically employed to achieve this. We suspected that the stochastic nature of the synaptic device prevents an over-fitting in this architecture and hence allows to generalize better. To test this hypothesis, we ran both FP32 and MCA training simulations using varying dropout factors while keeping the other hyperparameters to be the same. We scale both dropout rates (0.5 between the fully-connected and 0.25 after the pooling layers) with a scaling factor which takes the values 0.0, 0.2, 0.4, 0.8, 1.0, 1.2, and 1.4. The resulting maximal test accuracies are depicted in **Figure 4C**. As dropout rates are reduced, MCA training achieves higher test accuracies compared with FP32 training, indicating the inherent regularization achieved via MCA.

A.3.4. LSTM

The LSTM network trained using MCA for the task of character-level language modeling contains roughly 3.3 million parameters (**Figure 4D**). An LSTM cell takes as input a hidden state from the previous time step h^l_{t-1} and the training data x_t or the hidden state from the previous layer h^{l-1}_t and generates a new hidden state h^l_t and updates a cell state c^l using the weights w^l_f, w^l_i, w^l_g, w^l_o, for $l = 1, 2$ according to the following relations:

$$\begin{pmatrix} i_t \\ f_t \\ o_t \\ g_t \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ tanh \end{pmatrix} \left[W \begin{pmatrix} x_t \text{ or } h^{l-1}_t \\ h^l_{t-1} \end{pmatrix} + b^l \right] \tag{A1}$$

$$c^l_t = f_t \odot c^l_{t-1} + i_t \odot g_t \tag{A2}$$

$$h^l_t = o_t \odot tanh(c^l_t) \tag{A3}$$

where *sigmoid* (σ) and *tanh* are applied element-wise and $\odot$ is the element-wise multiplication. W is obtained by stacking w^l_f, w^l_i, w^l_g, w^l_o. i_t, f_t, o_t, g_t, h^l_t, and c^l_t are of dimension $n = 512$ and x_t is of dimension $m = 50$. The weight matrix, W, is of dimension $4n \times (n+m)$ in the first layer and $4n \times 2n$ in the second layer. b^l is a $4n$-dimensional bias vector. Dropout is applied to the non-recurrent connections (i.e., at the output of each LSTM cell) (Zaremba et al., 2014). The output of the second LSTM cell is fed through a fully-connected layer with 50 output neurons and then through a softmax activation unit.

The PTB dataset has 5.1 million characters for training, 400 k characters for validation, and 450 k character for testing. The vocabulary contains 50 different characters and the data is fed into the network as one-hot encoded vectors of dimension (50×1), without any embedding. Each vector has a single distinct location marked as 1 and the rest are all zeros.

The FP32 baseline and the MCA implementation of the LSTM network were trained using SGD with a batch size of 32, backpropagation-through-time steps of 100, and cross-entropy loss function. We used a learning rate of 2.0, which was dropped by a factor of 0.25 at epochs 40 and 80. The training performance is evaluated using the bits-per-character (BPC) metric, which is the average cross-entropy loss evaluated with base 2 logarithm. In the MCA implementation, the weights in all the layers are represented and trained using PCM device models organized in crossbar arrays (see **Supplementary Note 7**). The remaining gating operations are performed in the conventional digital domain. Conductance refresh was performed after every 11 batches in the MCA implementation.

A.3.5. Testing the Regularization Effect During Training of the LSTM

To study the regularization effect, the LSTM network was trained with different dropout rates, from 0.0 to 0.25 in steps of 0.05. The resulting minimal training and test BPC can be seen in **Figure 4F**. The performance on the test dataset was less sensitive to the dropout rate in MCA and it achieved lower BPC without any dropout compared to FP32 based training. This result is also consistent with previously reported studies of training LSTMs using memristive crossbars (Gokmen et al., 2018). Considering the rather low number of parameters in the network, the test BPC results compare favorably with the current state-of-the-art methods (Merity et al., 2018).

A.3.6. GAN

A multi-layer perceptron (MLP) implementation of GAN (Goodfellow et al., 2014) was employed with approximately 4 million trainable parameters. The generator has two hidden layers with ReLU activations and an output layer of sigmoid activation. The discriminator has two hidden layers with Maxout activations (maximum out of its five inputs) and a sigmoid output layer. Due to the large number of parameters in the discriminator, we used dropout regularization with dropout rates 20% and 30%, respectively after its first and second hidden layers. The GAN is trained using 50,000 images from the MNIST dataset and its performance is evaluated using 10,000 test images. G_{LOSS} and D_{LOSS}, which represent the loss functions minimized to train the generator and discriminator respectively, are as follows:

$$G_{\text{LOSS}} = \sum log(D(G(z)) \tag{A4}$$

$$D_{\text{LOSS}} = \sum (log(D(x)) + log(1 - D(G(z)))) \tag{A5}$$

where, x and z respectively are the MNIST images and the random input noise. $G(.)$ and $D(.)$ represent the generator and the discriminator networks and summation is over the training samples.

The training performance of the GAN is evaluated using the Frechet distance (FD), a metric that is robust even in the presence of common failure modes in GAN (Heusel et al., 2017; Lucic et al., 2017; Liu et al., 2018). FD makes use of features extracted from a particular layer in a CNN trained to classify the training dataset (Shor, 2017).

$$FD = ||\mu_X - \mu_G||_2^2 + Tr(C_X + C_G - 2(C_X \times C_G)^{1/2}) \tag{A6}$$

μ_X and μ_G denote the mean values of features computed for the train/test dataset and the generated dataset, respectively. C_X and C_G are covariance matrices of features computed for the train/test dataset and the generated dataset, respectively. Tr is the trace operator over a matrix and $||.||_2$ is the 2-norm operator.

The generator and the discriminator networks were trained using SGD with a minibatch size of 100, a learning rate of 0.1, and a momentum of 0.5 in FP32 baseline and MCA. An exponential learning rate decay with a decay rate of 0.996 was employed after every epoch until it reaches 10^{-6}. In the MCA implementation, the weights in all the layers are represented and trained using PCM device models organized in crossbar arrays (see **Supplementary Note 7** for more details on training). Conductance refresh was performed after every 11 batches in the MCA implementation.

A.3.7. Study of Batch Size and Optimizer for GAN

To study the sensitivity of training GANs to the mini-batch size and the choice of optimizer used for training, we trained FP32 (baseline) GAN implementation with two mini-batch sizes of 1 and 100 and with two different optimizer types: SGD without momentum and SGD with momentum while keeping all the other hyper-parameters the same. **Figure 4I** compares results in these cases. Non-unity batch size with momentum was necessary to train the GAN successfully in our case. In all the other cases the solution seems to diverge; the discriminator loss goes to zero and the generator loss diverges. Similar behavior is observed frequently in generative networks (Arjovsky et al., 2017).

A.4. Energy Estimation of MCA and Comparison

The energy efficiency of the MCA compared with a conventional 32-bit fixed-point digital design and a digital mixed-precision design for training was evaluated using respective ASIC implementations in 14LPP technology (see **Supplementary Note 8**). The three designs were customized to perform the training operations of a two-layer perceptron trained to classify MNIST handwritten digits. They also accommodated all the necessary SRAM memory on-chip, avoiding the cost of off-chip memory access. The network had 784 inputs, 250 hidden neurons, and 10 output neurons as in the training experiment presented in section 2.3. For simplicity of the ASIC design, we used ReLU activations for the hidden layer neurons and L2SVM (Tang, 2013) for the error computation at the output layer. The network was trained using SGD with a batch size of 1.

Cycle-accurate register transfer level (RTL) models of the 32-bit all-digital design, digital mixed-precision design, and the digital unit of the MCA were developed. A testbench infrastructure was then built to verify the correct behavior of the models using the Cadence NCsim simulator. Once the behavior was verified, the RTL models were synthesized in Samsung 14LPP technology using Cadence Genus Synthesis Solution software. The synthesized netlists were then imported to Cadence Innovus software where the netlists were subjected to backend physical design steps of placing, clock tree synthesis, and routing. At the end of routing, the post route netlists were exported with which post route simulations were carried out in NCsim simulator. During each simulation, an activity file is generated which contain toggle count data for all the nets in the netlist. The activity data along with the parasitics extracted from the netlist were used to perform an accurate power estimation on the post route design for each stage of operation (forward propagation, backward propagation, weight update), with the Innovus software. For power estimation, a supply voltage of 0.72 V was used in the designs while an operating frequency of 500 MHz was used to clock the fully digital designs and the digital unit of MCA. Power numbers were then converted to energy by multiplying by respective time windows. The energy estimations from the digital unit were combined with those from the computational memory to determine the overall performance of the MCA implementation. The computational memory unit of the MCA was designed separately with the necessary peripheral circuits to integrate it with the digital unit of the MCA. Note that most of the memristive device technologies including PCM are amenable to back end of line (BEOL) integration, thus enabling their integration with mainstream front end CMOS technology. This allows the crossbar array peripheral circuits to be designed in 14LPP. To support the device programming, a separate power supply line might be necessary. Larger currents where necessary can be supported by fabricating wider or a parallel combination of transistors. Operating at 2 GHz, the computational memory unit used 16-bit wide bus for data transfer, pulse width modulators to apply digital variables as analog voltages to the input of the crossbar array, and ADCs to read the resulting output currents. The time and energy consumption of the peripheral circuits were obtained from circuit simulations in 14LPP technology. The energy consumption of the analog computation was estimated assuming the average device conductance of 2.32 μS observed from the hardware training experiment. The average programming energy for the PCM conductance updates in the computational memory unit was estimated based on the SET and RESET (for weight refresh) pulse statistics from the training experiment. The device programming was executed in parallel to the weight update computation in the digital unit. In contrast to the hardware experiment, the weight refresh operation was distributed across training examples as opposed to periodically refreshing the whole array during training, which allowed it to be performed in parallel with the weight update computation in the digital unit (see **Supplementary Note 8**).

The details on the design of the three training architectures and the corresponding energy estimations can be found in **Supplementary Note 8**. A summary of energy and computing time for the forward and backward data propagations and the weight update stages for the designs are listed in **Table 1**. The energy and time are reported as average numbers for a single training example. Since the designs are operating at different precisions, throughput is reported as training examples processed per second. The baseline 32-bit implementation consumed 14.35 μJ with a throughput of 43k images per second. Computational memory enabled an average energy gain of 269 and an acceleration of 9.50 for the forward and backward propagation stages with respect to the 32-bit design. The low-precision implementation of the outer product led to reduced precision multipliers (3-bit compared to 32-bit) and sparse access to the 32-bit χ memory. These factors led to 139× improvement in energy consumption for the digital weight update accumulation stage in MCA, with 2130 non-zero updates to the χ memory being performed per training image. We verified via training simulations using the PCM model that the low-precision outer product optimization of the weight update could maintain comparable test accuracy as that from the PCM hardware training experiment (see **Supplementary Note 8**). Energy consumption due to the PCM programming in the computational memory was negligible compared with the energy spent in the digital unit of MCA for the weight update stage. Overall, the MCA consumed 83.2 nJ per image and achieved 495k images/s throughput. The digital mixed-precision design followed a similar architecture as that of the MCA. However, the computational memory was replaced by a multiply-accumulate unit for the data propagation stages, optimized to use 4-bit weights and 8-bit activations or errors. Note that activations and error vectors have additional bit-shift based scaling factors to represent their actual magnitude. The digital mixed-precision design had the same reduced precision weight update scheme as the MCA, and hence a similar energy efficiency for weight updates. However, for the data propagation stages, the 4-bit weights were obtained from the 32-bit weights read from the on-chip SRAM. The requirement to read a high-precision memory for the data propagations is avoided in the in-memory computing architecture. As a result, MCA maintained an energy efficiency gain of 85× during the data propagation stages, and 22× overall with respect to the digital mixed-precision design. The fully digital mixed-precision design consumed 1.87 μJ per training example with a throughput of 136 k images per second.