Deep Learning Acceleration Based on In-Memory Computing

E. Eleftheriou, M. Le Gallo, S.R. Nandakumar, C. Piveteau, I. Boybat, V. Joshi, R. Khaddam-Aljameh, M. Dazzi, I. Giannopoulos, G. Karunaratne, B. Kersting, M. Stanisavljevic, V. P. Jonnalagadda, N. Ioannou, K. Kourtis, P. A. Francese, A. Sebastian

Performing computations on conventional von Neumann computing systems results in a significant amount of data being moved back and forth between the physically separated memory and processing units. This costs time and energy, and constitutes an inherent performance bottleneck. In-memory computing is a novel non-von Neumann approach where certain computational tasks are performed in the memory itself. This is enabled by the physical attributes and state dynamics of memory devices, in particular resistance-based non-volatile memory technology. Several computational tasks such as logical operations, arithmetic operations and even certain machine learning tasks can be implemented in such a computational memory unit. In this paper we will first introduce the general notion of in-memory computing and then focus on mixed-precision deep learning training with in-memory computing. The efficacy of this new approach will be demonstrated by training the MNIST multilayer perceptron network achieving high accuracy. Moreover, we will show how the precision of in-memory computing can be further improved through architectural and device-level innovations. Finally, we will present system aspects, such as high-level system architecture, including core-to-core interconnect technologies, and high-level ideas and concepts of the software stack.

1 Introduction

Recent years have witnessed a tremendous explosion of data, which continues unabatedly: it is estimated that the digital universe is growing at a rate of about 60% per year. This abundance of data significantly improves our understanding of today's incredibly complex economies and societies. Moreover, it ushers in a new era of computing, viz. the cognitive or AI era, in which data is considered a new natural resource.

In today's computing systems based on the conventional von Neumann architecture, there are distinct memory and processing units. Performing computations results in a significant amount of data being moved back and forth between the physically separated memory and processing units. This costs time and energy, and constitutes an inherent performance bottleneck. Moreover, the memory unit itself, which typically comprises dynamic random-access memory (DRAM) for storing the information in the charge state of a capacitor, is volatile and consumes a large amount of energy. Thus, the classical von Neumann computing architecture poses serious challenges in terms of area and power consumption for tackling the high computational complexity and big data volume required by AI workloads. This has triggered research efforts to unravel and understand the highly efficient computational paradigm of the human brain, with the aim of creating brain-inspired computing systems. IBM, with its fully digital TrueNorth chip architecture, reached a key milestone in mimicking neural networks "in silico" [1]. In addition to fully digital approaches, analog and hybrid architectures are also being investigated.

Most recently, post-silicon nanoelectronic devices with resistive memory (memristive) properties are also finding applications beyond the realm of memory. It is becoming increasingly clear that for AI application, we need to transition to computing architectures in which memory and logic coexist in some form. Brain-inspired neuromorphic computing and the fascinating new area of in-memory computing or computational memory are two key non-von Neumann approaches being researched [2]. A critical requirement in these novel computing paradigms is a very-high-density, low-power, variable-state, programmable and non-volatile nanoscale memory device. Phase-change-memory (PCM) [3] based on chalcogenide phase-change materials, such as $Ge_2Sb_2Te_5$, is technologically one of the most mature memristive technologies and as such is well suited to address this need, owing to its multi-level storage capability and potential scalability. In in-memory computing, the physics of the nanoscale memory devices – as well as the organization of such devices in crossbar arrays – are exploited to perform certain computational tasks within the memory unit (see **Figure 1**) [4-9]. The essential idea is to treat memory not as a passive storage entity, but to exploit the physical attributes of the memory devices and thus realize computation exactly at the place where the data are stored. Several computational tasks such as logical operations, [10,11] arithmetic operations [12,13] and even certain machine learning tasks [14] can be implemented in such a computational memory unit. Specifically, crossbar arrays of PCM or other memristive devices can be used to store a matrix and perform analog matrix-vector multiplications at constant $O(1)$ time complexity without intermediate movement of data. This capability is well suited for solving complex optimization problems, such as compressed sensing and recovery [15,16]. For example, for a signal of size N, this method achieves a potential $O(N)$-fold complexity reduction compared with standard software for compressed sensing and recovery approaches. The same concept can also be

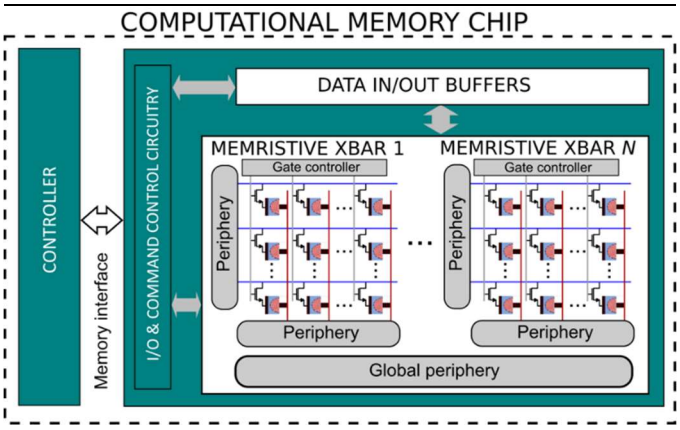

Figure 1 Potential architecture of a computational memory chip comprising multiple crossbar arrays of memristive devices performing computational tasks in place.

used to accelerate deep learning inference [9,17,18].

Besides the ability to perform logical operations, arithmetic operations and matrix–vector multiplications, another crucial property is that of realizing higher-level computational primitives by exploiting the rich dynamic behavior of the constituent devices. For example, the dynamic evolution of the conductance levels of those devices upon application of electrical signals can be used to perform in-place computing. Specifically, an algorithm to detect the temporal correlations between event-based data streams using a PCM-based computational memory has been presented in [14]. The reduction in complexity is from $O(N)$ to $O(k\log(N))$, where k is a small constant and N is the number of data streams. For example, it was shown that for 10 million data streams a 200x speed-up over 4 P100 GPUs could be achieved. Finally, in [19] it was shown that a feedback circuit in conjunction with a cross-point resistive memory can solve algebraic problems in one step.

Although these machine-learning applications are impressive, in order to reach the numerical accuracy typically required for data analytics and scientific computing, limitations arising from device variability and non-ideal device characteristics need to be addressed. Thus, the concept of mixed-precision in-memory computing – which combines a conventional high-precision von Neumann machine with a computational memory unit – was introduced in [20]. In this hybrid system, the computational memory unit performs the bulk of a computational task, whereas the von Neumann machine iteratively improves or refines the accuracy of the solution. The system therefore benefits from both the high precision of digital computing and the energy/areal efficiency of in-memory computing. The efficacy of this approach was experimentally demonstrated by accurately solving systems of linear equations [20].

The application of the mixed-precision in-memory computing framework to deep neural networks, which is the main focus of this paper, was proposed in [21-23]. This architecture combines a computational memory unit for storing the synaptic weights with a digital processing unit and an additional memory unit that stores the accumulated weight updates in high precision. Specifically, the expensive matrix-vector multiplication operations for forward and backward passes can be performed in place using a non-von Neumann unit, whereas the weight updates — which are also expensive — are accumulated in high precision in the digital unit. Subsequently, the weights are updated in place by exploiting the accumulative behavior. We will show that the new architecture delivers classification accuracies comparable to those of floating-point implementations without being constrained by challenges associated with the non-ideal weight update characteristics of emerging resistive memories.

The rest of the paper is organized as follows. In Section 2 we will detail the mixed-precision architecture and how it works. Moreover, we will also briefly describe the advantages of the approach as already demonstrated in [21,22], i.e., robustness to stochasticity, nonlinearity, asymmetry, etc.. In Section 3 we will first give a brief presentation of PCM technology and the experimental platform and then briefly present the experimental characterization result focusing on the quantitative match between model and experiment. The main theme of this section will be evaluating the training efficacy of the mixed-precision architecture under realistic PCM behavior. Section 4 will focus on improvements in performance, accuracy and reliability by introducing multi-PCM-cell synapses. We will also discuss the importance of the concept of projected memory to address the critical issue of sensitivity temperature variations and also to increase the precision of matrix-vector multiplication. Section 5 will focus on system aspects, such as high-level system architecture, including core-to-core interconnect technologies, and high-level ideas and concepts of the software stack. The last section will be the conclusions, summarizing the main results of the paper and also providing a brief outlook.

2 Mixed-precision architecture for deep learning

Deep neural networks (DNN) have many layers of neurons interconnected using adaptable weights. The DNNs are trained to perform a desired task traditionally via a supervised learning algorithm called back-propagation. The training involves three stages. During forward propagation, training data is propagated through the layers of neurons and the weighted inter-connections to determine the network response. During backward propagation, a cost function is determined based on the observed network output and its desired response. The gradient of the cost function with respect to the weighted sum in the output layer is back propagated through the weight layers to determine the corresponding gradients for all the weight layers. During the weight update stage, the neuron responses from the forward propagation and the gradients from the backward propagation are used to determine the desired weight updates. A crossbar array of analog memory devices could perform the matrix-vector multiplications during the forward and backward propagation stages. However, to train the DNNs, the device conductance values need to be adjusted with high-resolution, i.e., 10 or more bits, which presents a major challenge considering the physical nature of the nanoscale memory devices. The gradual conductance changes in the analog memory device generally involve an atomic rearrangement in a

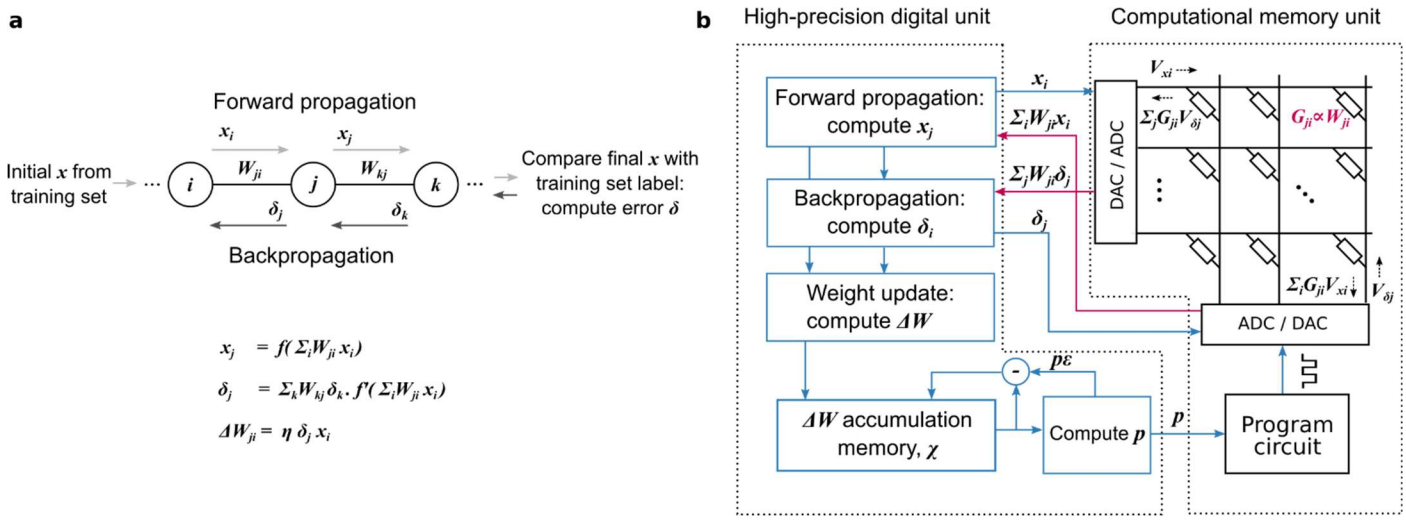

Figure 2 a. Basic computations for training a neural network. W_{ji} is the connection strength between a neuron in layer i to a neuron in layer j. The neuron activations x_i are multiplied by the connection strength and accumulated to determine the net input to the next layer neurons. A non-linear function f is applied over the weighted sum to determine the next layer neuron activations x_j. Likewise, the gradients from the layer j, δ_j is backpropagated through the weighted connections to determine the gradients at the preceding layer i, δ_i. The weight update $\boldsymbol{\Delta W_{ji}}$ is determined as a product of x_i and δ_j scaled by a suitable learning rate $\boldsymbol{\eta}$. **b.** Architecture of the computational memory based mixed-precision system for deep learning. The computational memory unit implements the DNN weighted connections using the conductance values of analog memory devices in a crossbar array. The crossbar arrays perform the weighted summation during the forward inference and backpropagation. The high precision digital unit determines weight updates, ΔW, based on the results from the computational memory unit. The updates are accumulated in high precision in χ. The number of programming pulses, p, to update corresponding devices are determined by rounding χ/ε towards zero, where ε represents the average device update granularity. After each conductance update in the computational memory, $p\varepsilon$ is subtracted from χ. Adapted from [21].

nanometric volume, which make them of limited precision, stochastic, and asymmetric. Accumulating gradients in high precision has been observed to tolerate limited-precision weights during training [24-28]. Inspired by this and our recent mixed-precision architecture [20], we developed a mixed-precision computational memory architecture for deep learning. Here, analog memory arrays are used for fast matrix-vector multiplications. The limited-precision weight updates are compensated by high-precision weight accumulations in a digital unit, and conductance updates are performed using sporadic and blind programming pulses [21].

The basic computations for training a neural network and the mixed-precision in-memory computing architecture for training DNNs are shown in **Figure 2**a,b, respectively. The mixed-precision architecture is comprised of a computational memory unit which has several memristive crossbar arrays storing the DNN weight values, and a high-precision digital computing unit. A weight, W_{ji}, in any layer of a DNN (Figure 2a) connecting neuron i to the next-layer neuron j is mapped to a conductance value G_{ji} using one or more analog memory devices in the computational memory. For the forward propagation, the neuron activations, x_i, are applied as corresponding voltage pulses, V_{xi}, to the crossbar rows. The resulting currents, which are proportional to the individual synaptic conductance values, will accumulate along the columns of the crossbar array i.e., $I_j = \sum_i G_{ji} V_{xi}$, and will correspond to $\sum_i W_{ji} x_i$. These currents, after sensing and digitization, using analog-to-digital converters (ADCs), become the input for the next-layer neurons. The same crossbar array can also be used to perform the matrix-vector multiplication during the backpropagation through the same layer. In this case, the errors δ_j to be backpropagated are applied as voltages $V_{\delta j}$ to the columns of the crossbar array, and the total current obtained along the rows represents the weighted sum $\sum_j W_{ji}\delta_j$, which can be used to determine the gradients for the neurons in the preceding layer.

The desired update for any weight layer is determined as an outer product of the pre-neuron activations and error at the post neuron pre-activations, i.e. $\Delta W_{ji} = \eta \delta_j x_i$, where η is a suitably chosen learning rate. The magnitudes of the weight updates are often many orders magnitude smaller compared to the corresponding weight values, while the update precision offered by the analog memory devices used to store them is only a few bits. This presents significant challenges to directly programming the updates to the devices. As a result, in the mixed-precision training architecture, we accumulate these small updates in a high-precision variable χ in the digital unit. When the weight updates accumulated over several training instances becomes comparable to the analog memory update precision, corresponding synaptic device conductance is modified. Let ε be the average change in the weight that can be reliably programmed to the device. Then the number of programming pulses p to be applied is determined by rounding χ/ε towards zero. The sign of p determines if the conductance is to be potentiated or depressed. We chose to use a blind device programming scheme, which does not verify if the resulting

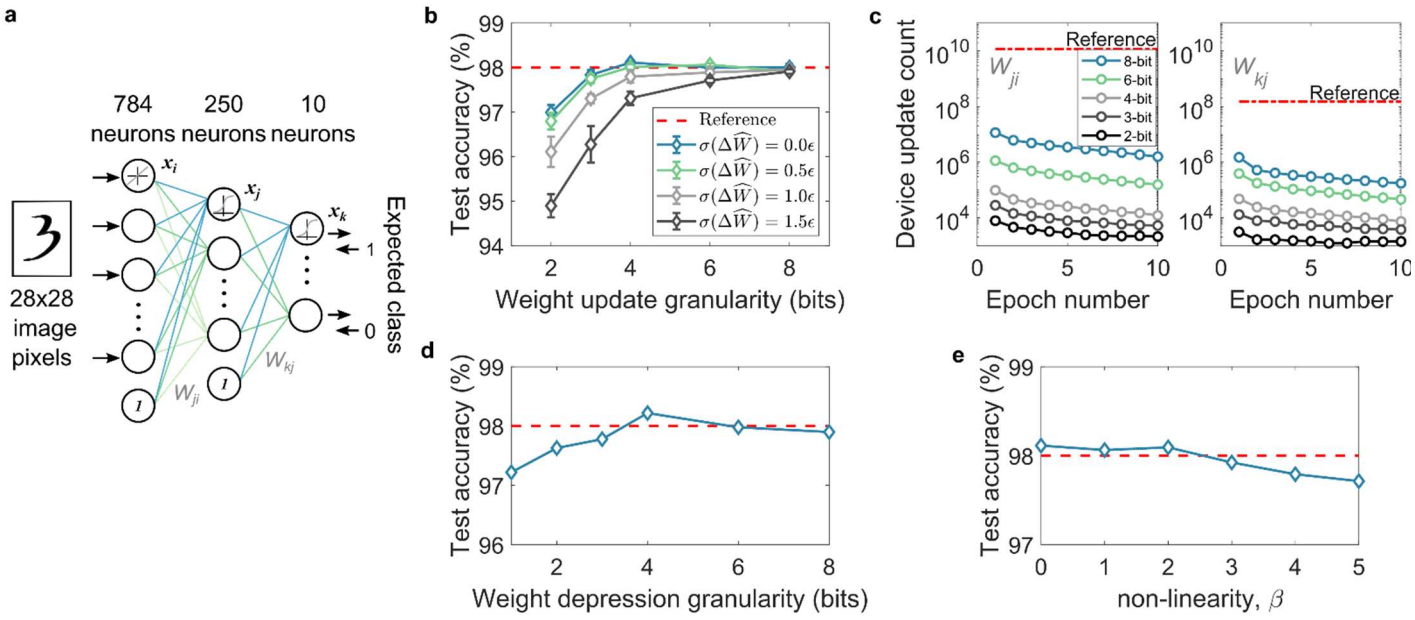

Figure 3 a. A neural network for classifying handwritten digits from the MNIST dataset. **b.** Linear devices with symmetric potentiation and depression granularity are assumed as computational memory elements. The standard deviation of the weight-update randomness, $\sigma(\Delta\hat{W})$, is taken as a multiple of the weight-update granularity, ε. **c.** The number of devices programmed per epoch for different values of ε. **d.** The effect of asymmetric conductance update. 8-bit resolution is assumed for the weight increment and weight depression resolution is varied. **e.** The test accuracy as a function of the weight update non-linearity. β=0 corresponds to linear update within the weight range [-1, 1] and β=5 corresponds to nearly an abrupt transition between the weight range boundaries. Adapted from [22].

conductance change matches the desired change, for efficiency. Once the weights stored in the analog memory are updated, $p\varepsilon$ is subtracted from χ. Note that, the device programming is often highly stochastic, non-linear, and asymmetric. However, mixed-precision training appears to compensate for these erroneous updates during subsequent training processes and achieve high classification accuracies [21].

The sensitivity of the mixed-precision architecture to different device non-idealities was evaluated via simulations by training a two-layer neural network to classify handwritten digits from the MNIST dataset, as shown in **Figure 3**a [21,22]. The number of neurons in the input, the hidden and the output layer is 784, 250, and 10, respectively and the hidden and the output neurons are sigmoid. The network weights were implemented using a computational memory array whose weight updates are assumed to exhibit non-ideal behavior. The network was trained using 60,000 gray-scale images of size 28×28 and its classification capacity was evaluated using a similar but disjoint set of 10,000 test images. For the test accuracies in Figure 3b, the memory device updates were linear with an update resolution of $\varepsilon = 2/(2^n\text{-}2)$ where the bit resolution n was varied between 2 to 8 within a hypothetical conductance range of [-1 1]. It was found that a 4-bit resolution was sufficient to attain a performance comparable to that of the 64-bit floating-point reference.

In this mixed-precision scheme, weight update accumulation can reduce the number of required device programming instances by more than two orders of magnitude, as smaller updates are combined and applied together to the device. As can be seen in Figure 3c, the device updates become sparser as the weight update granularity, ε, becomes larger. The weight update accumulation schemes enable the architecture to be more tolerant to device programming noise as well. Previous studies have indicated the need for highly symmetric weight updates with less than 2% error tolerance [29]. As a result, directly applying the weight updates to the non-ideal analog memory devices leads to significant loss in accuracy [30]. In the mixed-precision architecture, accumulating the weight updates in high-precision in the digital memory and sporadically applying them to the low-precision synaptic devices significantly relaxes the requirements of these nano-scale memory devices in terms of symmetry, precision and endurance. For example, the mixed-precision training architecture allows for different update thresholds for incrementing and decrementing the conductance values, depending on the device asymmetry characteristics, incurring negligible loss in training performance (Figure 3d). Finally, the mixed-precision training architecture is also tolerant to non-linear conductance updates when trained using average update resolution as the programming threshold (Figure 3e).

The computational efficiency of the mixed-precision in-memory training architecture, compared to conventional deep learning training approaches, arises from storing the synaptic weights in the conductance states of nanoscale non-volatile memory devises organized in crossbar arrays and performing the expensive weighted summations in place in a non-von Neumann manner. At the other end of the spectrum lies the in-

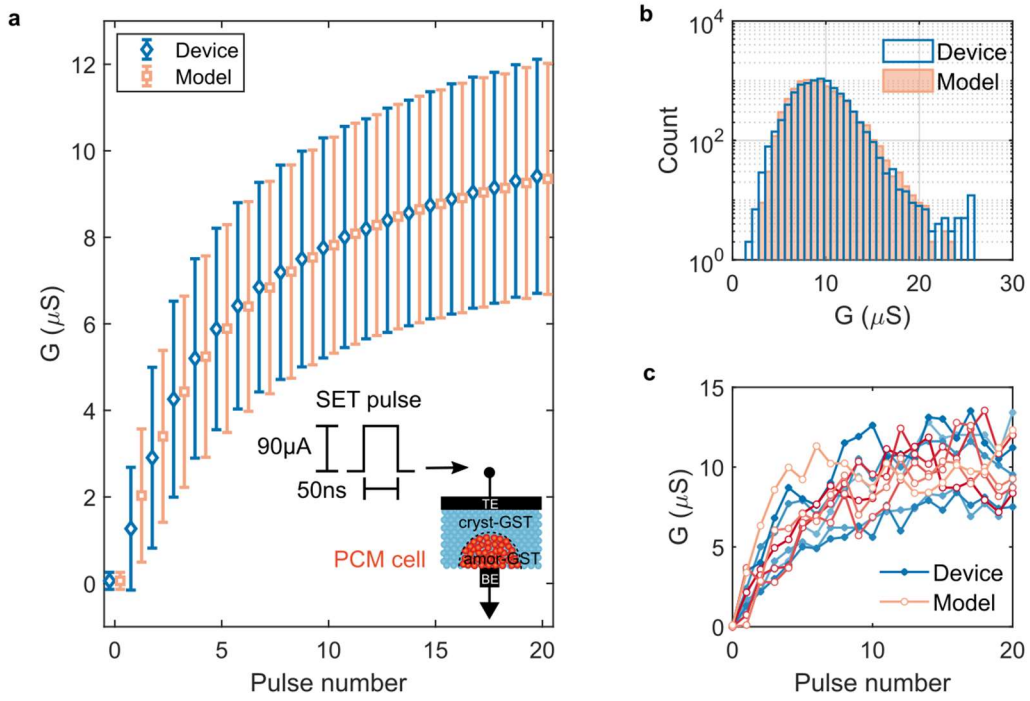

Figure 4 a. The statistics of conductance evolution from the devices and the corresponding model in response to repeated application of SET pulses of amplitude 90 μA and duration 50 ns. **b.** The distribution of conductance from the device and model after 20 SET programming pulses. **c.** Conductance evolution from individual instances of devices and model during the programming sequence. Adapted from [37].

memory acceleration with in-place updates, where all the stages of the training, including the forward and backward data propagation and the weight update stages are implemented in the computational memory array [29-31]. These schemes use a fully parallel conductance update approach by overlapping pulses from the pre- and post-synaptic neuron layers. However, this could often lead to stringent requirements from the non-volatile memory devices such as 10-bit granularity and symmetric updates. Thus, the implementation of gradient-descent-based training using large batch sizes and optimization schemes based on momentum and ADAM are quite challenging. Implementing the weight update stage in the high-precision digital unit allows our mixed-precision training architecture to exploit the efficiency of computational memory while maintaining the flexibility of the high-precision training systems [32].

3 Training simulation using PCM model

We also evaluated the mixed-precision training architecture using models that reliably capture the phase-change memory (PCM) programming statistics. PCM is one of the most advanced non-volatile memory technologies that has already found applications in storage-class memory [3], computational memory [14,15], and neuromorphic computing [2,33,34]. The memory device consists of a chalcogenide material sandwiched between two electrodes. As fabricated, the chalcogenide is in crystalline state and its phase can be altered by suitable melt-quench processes. The resistivity of the crystalline and amorphous phase of the material differs by more than two orders of magnitude and this allows the device to store information in its relative phase configuration. By passing a sufficiently large current (RESET pulse) through a relatively narrow bottom electrode, an amorphous region similar to the mushroom structure in **Figure 4**a can be created around the bottom electrode. The amorphous region blocks the conductance path between the electrodes in this phase configuration and the device will be in a high resistance state. The amorphous region can be progressively crystallized by increasing the device temperature to crystallization regime via current pulses of suitable amplitude (SET pulse). The ability to gradually modulate the device conductance by a sequence of SET pulses makes PCM a suitable candidate to implement the weight adaptation dynamics of DNNs on the chip.

We characterize doped $Ge_2Sb_2Te_5$ (GST)-based PCM devices from a prototype chip fabricated in 90 nm CMOS technology [15]. The array consists of a matrix of 512 word lines and 2,048 bit lines connected in a crossbar configuration. Each crosspoint consists of a PCM device in series with an access transistor. The cumulative conductance update behavior of the PCM is measured by initializing 10,000 devices to a conductance distribution around 0.06 μS and applying a sequence of 20 SET pulses of 90 μA amplitude and 50 ns duration. The devices show a non-linear saturating conductance update behavior (Figure 4a). The significant randomness observed in the

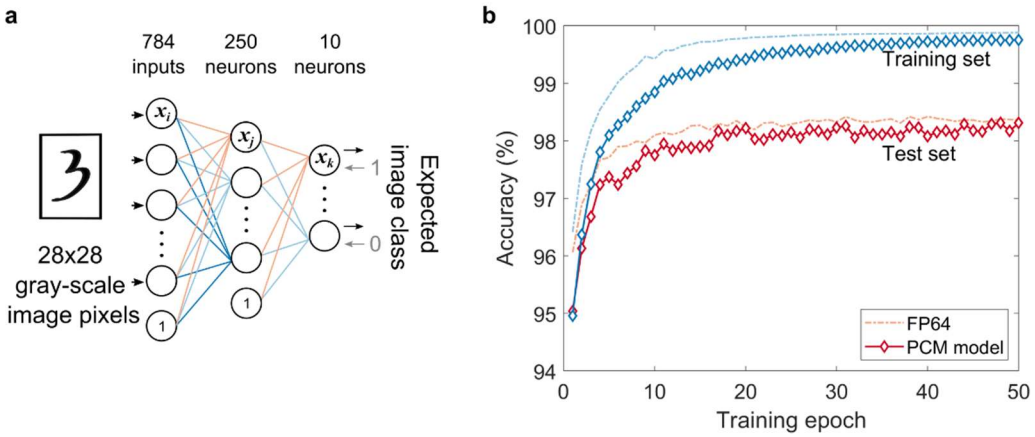

Figure 5 a. Network structure used for the mixed-precision in-memory training, based on a PCM model, for MNIST digit classification. Each weight W in the network is realized as the difference in conductance values of two PCM cells, G_p and G_n. **b.** Classification accuracies on the training and test set from the mixed-precision training simulation. The maximum test set accuracy, 98.31%, is within 0.11% of that obtained in the floating-point (FP64) software training. Adapted from [39].

conductance update behavior is a combination of inter and intra device variability. The inherent randomness associated with the crystallization process could be the significant contributing factor here [35, 36]. We analyzed the mean and standard deviation of conductance update from each programming pulse in a state-dependent manner. Using a combination of simple mathematical formulations, we were able to develop a phenomenological model that capture the conductance update statistics remarkably well [37]. In addition to the overall conductance distribution after each programming event (Figure 4b), the model also captures the individual device conductance evolution behavior reasonably well (Figure 4c).

The statistical model was used to emulate the conductance update behavior of a two-layer perceptron performing handwritten digit classification (**Figure 5**a). Each of the 198,760 weights in the network, W, is assumed to be realized using two PCM devices in a differential configuration, i.e., $W = \beta(G_p - G_n)$ for a scalar constant β. The model instances were initialized to a conductance distribution of 1.6 μS mean and 0.83 μS standard deviation. The conductance of the differential synapse in the range [-8 μS, 8 μS] was linearly mapped to [-1, 1] in the weight domain. The network was trained using the mixed-precision in-memory computing architecture. That means, the model conductance values were used to perform the matrix-vector multiplications for the forward and backward data propagation stages. Inputs and outputs of the computational memory arrays were quantized to 8-bits resolution assuming 8-bit analog-digital converters at the periphery. The weight updates were accumulated in χ memory and when the magnitude of χ exceeds ε, corresponding to a 0.77 μS conductance change, the device conductance values were updated using the model in a state-dependent manner. G_p is updated for potentiation and G_n is updated for depression. Due to accumulate and program scheme, the average number of devices updated after each training example was less than one, indicating that the actual device programming overhead will be negligible. To avoid conductance saturation from repeated SET pulse updates in the differential configuration, saturated device states were reprogrammed to the conductance difference of the pair every 100 training images.

The network was trained using 60,000 training images from the MNIST dataset for 50 epochs. We used stochastic gradient descent with unit sized training batch and a fixed learning rate of 0.2. The classification performance of the network on the training set and on a disjoint test set of 10,000 images are plotted as a function of training epoch is shown in Figure 5b. The network achieved a maximum test set accuracy of 98.31% under the mixed precision in-memory computing training compared to the 98.42% in the corresponding high-precision training. An online demonstration of the simulator can be found in [38]. The high-precision comparable classification accuracy from the PCM based neural network suggests that the computational memory based mixed-precision training architecture can potentially train realistic analog memory devices to find solutions to complex deep learning problems.

4 Improving the memory: architectural and device-level innovations

The presented simulation results of neural networks using the mixed-precision in-memory computing architecture approach have demonstrated the computational capabilities of today's resistive memory technology. However, enhancing the performance of the devices is crucial for increasing the range of targeted applications and building next-generation of computational memory-based systems. For example, PCM devices exhibit a limited dynamic range, which can be covered upon the application of only a few electrical pulses. In addition, the conductance response is highly nonlinear and stochastic. Moreover, additional technology-specific device behavior such as the conductance response asymmetry, temporal evolution of conductance values, (i.e. conductance drift), limited endurance

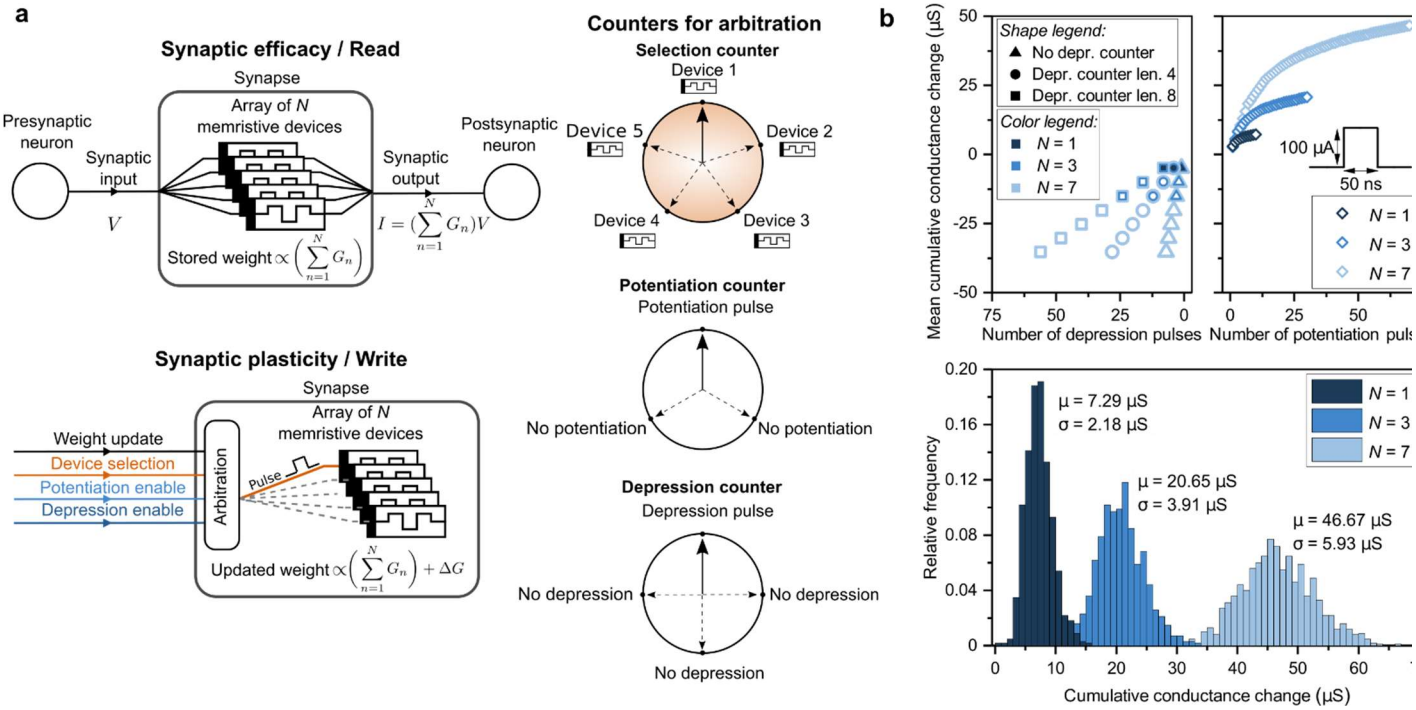

Figure 6 a. The multi-PCM synaptic architecture. The net synaptic weight is represented by the combined conductance of *N* PCM devices. To realize synaptic efficacy (read operation), a read voltage signal, V, is applied to all devices. The resulting current flowing through each device is summed up to generate the synaptic output. To capture synaptic plasticity (write operation), only one of the devices is selected with the help of a selection counter and programmed at any instance of synaptic update. In addition to the selection counter, independent potentiation and depression counters can serve to control the frequency of the potentiation or depression events. **b.** Using multi-PCM synapses helps increasing the dynamic range and improving the asymmetric and nonlinear conductance response (upper) and stochasticity (lower) of PCM. Measurements are based on 1,000 90 nm GST mushroom multi-PCM synapses. Adapted from [41].

(approx. 10^9 to 10^{12}), the frequency-dependent noise arising from the amorphous phase and temperature-dependent conductance variations pose significant challenges for PCM. These device non-idealities have been shown to have a detrimental impact on computational memory-based systems where resistive memory serves as the computational primitive [30,40-42].

One path to address some of these challenges is advances in synaptic-cell architectures. Using multiple PCM devices as a single synaptic unit has been shown to improve the conductance change granularity, nonlinearity, asymmetry, and stochasticity of PCM as well as the conductance drift for both neuromorphic and in-memory computing [40,42]. Note that such an architectural advancement is applicable not only to PCM but to a wide range of other resistive memory technologies [43]. A second path for improved memory performance is advances in materials science and device technology. The projected PCM concept aims to decouple the device read out from the electrical properties of the amorphous phase thus significantly reducing the drift, noise and temperature-sensitivity [44-46]. For the remainder of this section, the multi-PCM synaptic architecture and projected-PCM devices will be described in more detail.

4.1 Architectural level innovations: multi-PCM synaptic architecture

In a multi-PCM synapse, the synaptic weight is represented by the combined conductance of *N* devices, see **Figure 6**a. By using multiple devices to represent a synaptic weight, the overall achievable conductance range, i.e., $G_{max} - G_{min}$, and resolution of the synapse are increased. When reading the synapse, the individual device conductances are read and summed. For programming the synapse, only one out of *N* PCM devices is selected and updated at a time. This selection is done with a counter-based arbitration scheme, where the value of a counter indicates which device from the synapse is to be updated at any particular instance. A single selection counter is used for an entire array of synapses and is incremented after every device update. In addition to the global selection counter, additional independent counters, such as a potentiation counter or a depression counter, can be used to modulate the frequency of weight increase (potentiation) and decrease (depression) respectively. These secondary counters can be particularly useful for devices with asymmetric conductance response as well as devices with large conductance change. These additional counters are incremented after every device update and the device update is enabled only when their value is 1. This implies that only one out of *m* updates is applied if *m* is the maximum value or length of these counters.

Experimental characterizations of multi-PCM synapses comprising 1, 3, and 7 PCM devices per synapse shows, Figure

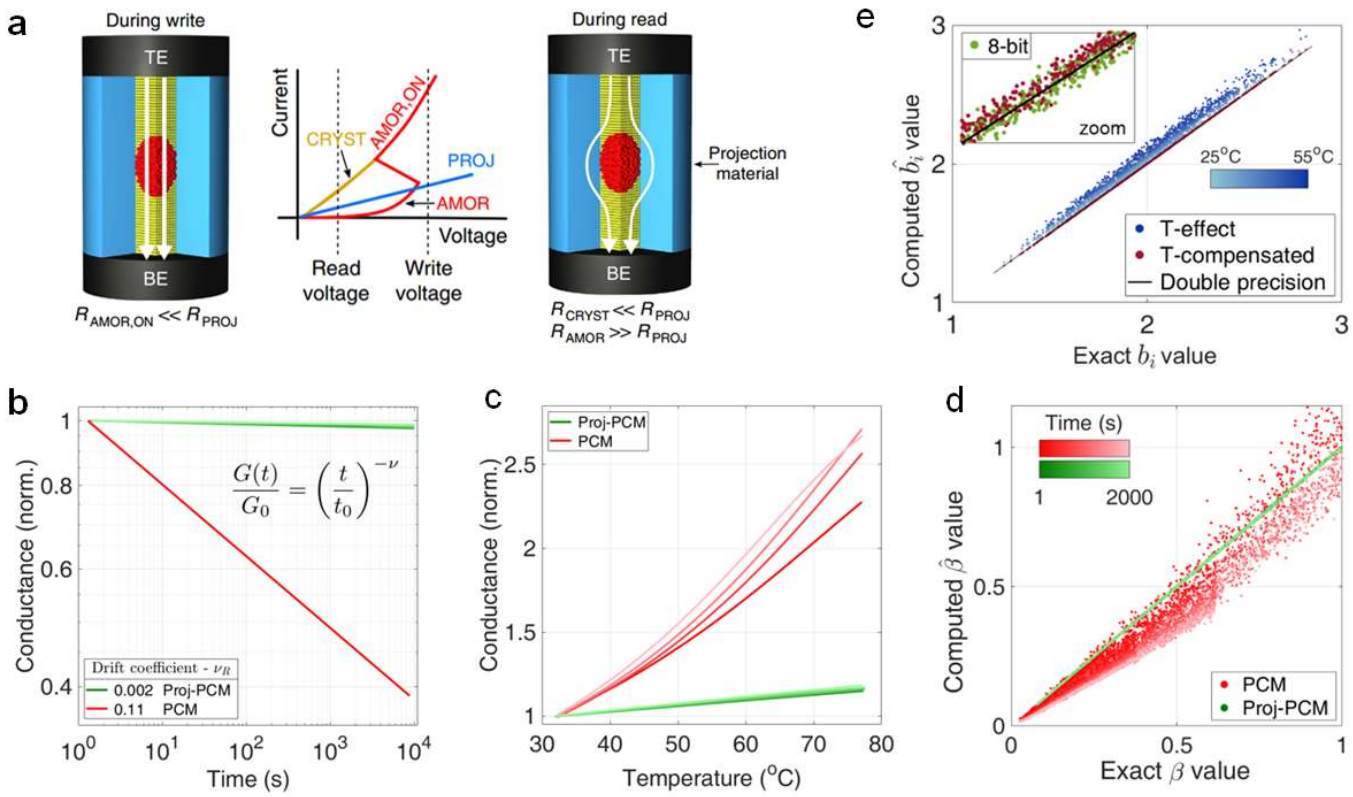

Figure 7 a. The concept of projected memory: The non-linear IV-characteristic of the amorphous phase ensures that during "write" most of the current flows through the phase-change segment resulting in phase transition, but during "read" it bypasses the highly resistive amorphous phase and flows through the projection material. **b.** Normalized conductance versus time for different programmed states in projected-PCM, compared to conventional PCM. The drift coefficient ($\boldsymbol{\nu}$) is determined by a power-law fit and was measured 50x reduced. **c.** Normalized conductance versus temperature for different annealed states in PCM and projected-PCM. **d.** 20,000 scalar multiplication results $\widehat{\boldsymbol{\beta}}$ against exact ones $\boldsymbol{\beta}$ on conventional and projected PCM devices. The color gradient shows the effect of resistance drift. Drift, 1/f noise and non-Ohmic behavior on PCM causes poor precision, as opposed to projected-PCM. **e.** Experimental emulation of 2000 matrix-vector multiplications employing 12 projected-PCM devices arranged in a 4×3 virtual crossbar configuration. Precision loss of results $\widehat{\boldsymbol{b}}_{\boldsymbol{i}}$ compared to the exact ones $\boldsymbol{b}_{\boldsymbol{i}}$ at elevated temperatures is recovered by the compensation scheme and is equivalent to the 8-bit fixed point arithmetic. Adapted from [45,46].

6b, that the conductance range scales linearly with the number of devices per synapse. Furthermore, a linear conductance change can be obtained over an extended range of pulses. With multiple devices, the challenge of an asymmetric conductance response can be partially mitigated. Moreover, the variance of the overall conductance change scales linearly with the number of devices per synapse, leading to an increase in the resolution of the synapse when using more devices. The concept is shown to significantly improve accuracies for 2nd generation (ANNs) and 3rd generation (SNNs) of neural networks through simulations and experiments [41].

A key advantage of the proposed multi-PCM synapse is its crossbar compatibility. In addition, the architecture does not bring a significant energy overhead since PCM devices can be read with low energy (1 - 100 fJ per device) [15] and only one device is programmed per update as in a conventional synapse. Although the area usage increases, multi-PCM synapses could still be very area-efficient since it is reported that PCM devices could be scaled to very small dimensions of a few tens of nanometers [3]. The proposed architecture also offers several advantages in terms of reliability. The other constituent devices of a synapse could compensate for the occasional device failure. Moreover, each device in a synapse gets programmed less frequently than if a single device were used, which effectively increases the overall lifetime of a multi-PCM synapse. The potentiation and depression counters reduce the effective number of programming operations of a synapse, further improving endurance-related issues.

4.2 Device-level innovations: projected phase change memory

Scalar multiply operations can be performed using PCM by mapping the one variable proportionally to a read voltage and the other into a conductance state. Due to Ohm's law, one can obtain the result from the read current. In addition, if devices are organized in a crossbar configuration, one can multiply a matrix with a vector by invoking the Kirchhoff's current summation rule. These scalar and matrix-vector multiplication operations

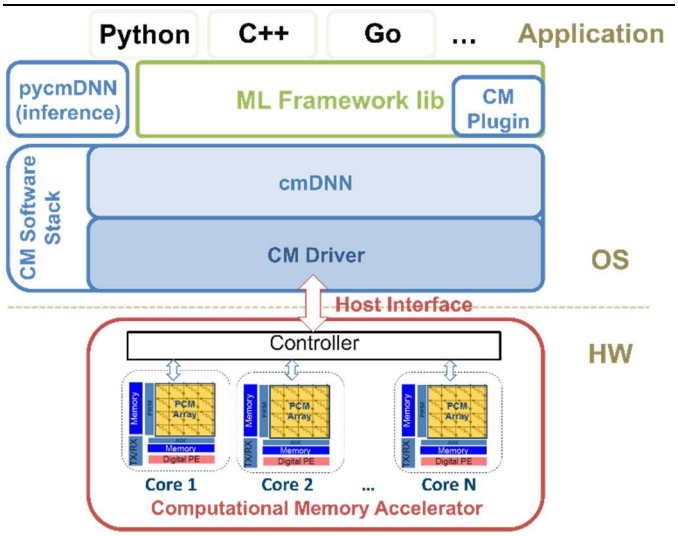

Figure 8 Overview of the system integration for an in-memory computing DNN accelerator.

are critical for in-memory and neuromorphic computing and the precision of these operations is strongly determined by the conductance variations associated with the amorphous phase of these devices. We introduced a device-level solution, namely the projected memory, to increase the multiplication precision and address the critical issue of sensitivity to temperature variations [44-46].

In a projected memory device, the essential idea is to design the device such that the physical mechanism of information storage is decoupled from the information-retrieval process. The projected phase-change memory device consists of a non-insulating projection segment in parallel to the phase-change segment. By exploiting the highly non-linear IV (current-voltage) characteristics of the amorphous phase, it is ensured that the projection segment has minimal impact on the operation of the device during the write process. However, during read, the programmed state's conductance is determined by the fraction of the projection segment that appears parallel to the amorphous phase-change segment as seen in **Figure 7**a.

Conductance drift [47] has a detrimental effect on maintaining reliably the programmed values. We have shown, Figure 7b, that in projected-PCM devices conductance drift can be reduced 50-fold. Another key challenge is that the activation energy of the thermally activated electrical transport tends to fluctuate for different programmed states. This varying exponential temperature dependence of conductance hinders temperature compensation schemes on an array level. Projected-PCM devices exhibit a linear, homogeneous and substantially weaker temperature dependence as is shown in Figure 7c. In addition, the read-current noise was measured 4 orders of magnitude lower in projected-PCM devices compared to conventional PCM. Collectively, all these performance enhancements allow us to execute in-memory operations with precision that has never been observed so far in any of the resistive technologies. Using projected PCM devices, it is possible to achieve a scalar multiply operation with precision equivalent to 8-bit fixed point arithmetic at room temperature, a remarkable improvement over conventional PCM, see Figure 7d.

One particularly tantalizing prospect is to compensate for the temperature-dependent conductance variations at a crossbar array level. Projected-PCM devices with their well-defined and state-independent temperature dependence of electrical transport are much more amenable to an effective compensation scheme. This is devised by multiplying the read current with a single-variable equation that describes the temperature dependence of the projection material, that is $f(T) = 1 + \alpha_p(T\text{-}T_0)$. The efficacy of this method to retain the 8-bit-equivalent precision at elevated temperatures is experimentally proven, as can be seen in Figure 7e.

5 System integration

In this section we discuss the system integration aspects of the proposed DNN accelerator. Firstly, we present an overview of the end-to-end system. We then discuss the high-level system architecture of the hardware accelerator itself, including on-chip core-to-core interconnect, and other hardware design choices and considerations. Then we detail the software stack that spans from user-level application all the way to the low-level driver that directly controls the accelerator. Finally, we discuss the hardware acceleration and efficiency gain attainable from the computational memory based mixed-precision architecture. At the core of the software stack is a DNN compiler that translates models into optimized operations for the accelerator. **Figure 8** depicts an overview of the different system integration components. Besides the hardware accelerator itself, there are three software components: the computational memory OS driver, the computational memory compiler and a library that allows user applications to directly perform inference or training on the hardware accelerator without having to deal with its low-level details. The user should be able to train a DNN architecture using an existing DNN framework (e.g., Tensorflow [48]) and the software stack should be responsible to utilize the DNN accelerator hardware and improve training time. Similarly, for inference, the user should be able to provide a pre-trained model in a standard format (e.g., ONNX [49]) and test data to run inference on. The software stack is responsible to transparently compile the model into optimized operations and routing and orchestrate model and data movement to and from the accelerator both for training and inference.

5.1 Hardware accelerator

As discussed in the previous sections the hardware is an array of computational memory-based processing units, made up of a crossbar array of PCM devices, and a digital processing element handling the activation functions and batch normalization. Compared to all-digital implementations, in memory computing is more amenable for highly pipelined dataflows, which make full use of the physically instantiated neurons and bear significant advantages in terms of throughput, latency and power consumption. We explore their use in the execution of convolutional neural networks, which feature the most diverse set of connectivity and therefore are the hardest to physically map on the hardware. In this case, the suitability of in-memory computing for pipelining techniques is further emphasized by

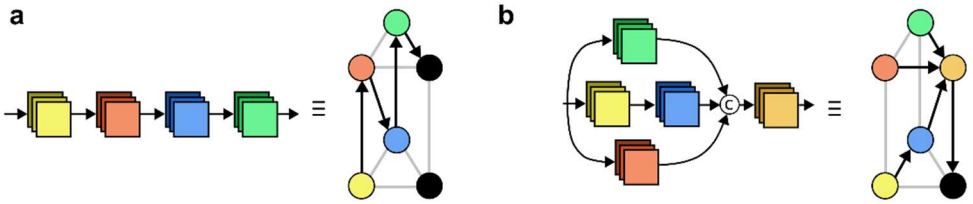

Figure 9 a. Example of mapping for a four-layer feedforward network and **b.** a five-layer Inception-style network using our proposed topology. In b, vertex C represents the concatenation operation. Adapted from [54].

the time complexity of the convolution operation. In all-digital accelerators, convolutions at different layers, with a different number of input and output channels, are executed with different latency. This is detrimental to a pipelined execution, since layer latency is not uniform across the network and the pipeline would ultimately move with the latency of the most computationally intensive layer. Conversely, the computational memory architecture maps all the filter kernels in a convolution layer to a single crossbar array (Figure 6) and hence generates all the filter responses corresponding to an input patch in parallel irrespective of the number of filters. The next layer can start the convolution as soon as an area corresponding to next layer filter size is processed from the current layer.

Given the entirety of physically static, temporally uniform convolutional layers for a certain CNN architecture, the challenge is transferred to providing a communication fabric that can efficiently move activations from one computational memory unit to another. In order to maintain the synchronicity of the pipeline, the fabric must exactly mirror the connectivity of the CNN architecture. This task is crucial in the design, since the overall pipeline latency goes with the latency of the slowest connection and, as described above, given that the computational time per layer is uniform throughout the network, communication time between layers should also be uniform.

It is then evident that any endeavour that adapts communication infrastructures from digital systems would be unsuitable for pipelined dataflows. i.e., taking a common 2D mesh as an example, except for the pre-2014 path-connected CNNs, the mapping of modern CNNs would ill-fit the hardware connections, i.e., certain data movements would require multiple hops, thus slowing down the entire pipeline.

This problem formulation and principles apply to both forward inference and backpropagation. However, in the latter case the hardware should provide the capabilities of performing the transpose read on the crossbar arrays and also support the transmission of data on the links bidirectionally. In this case, in order to support backpropagation, the hardware communication infrastructure must allow data movement that exactly matches the network connectivity in the same fashion as for the forward dataflow.

In general terms, it has to be assessed whether the problem of designing such topology is well posed. For example, a fixed, hardware communication fabric cannot flexibly mirror the connectivity of several state-of-the-art CNNs if these are structured in an unsystematic, inconsistent way. Nevertheless, considering CNNs in terms of their graph representation, it is apparent their topologies are not an arbitrary collection of vertices and edges. That is their connectivity is regular and symmetric, so much so that one specific type of connectivity can be considered archetypal of one category of CNNs, and any basic narrative on the evolution of CNNs structures is related with equal effectiveness as a narrative on the evolution of the connectivities used therein.

We recognize four major shifts in the CNN architecture paradigm, which match shifts in their connectivity. Path connectivity featured from the dawn of CNNs [50] throughout their high in popularity in 2012, Inception-style [51] connectivity with multiple parallel connections regularly coalescing into a concatenation of the feature maps, residual [52] connectivity with residual connections, and ultimately DenseNet-style [53] connectivity.

Thus, given a clear display of the topologies for which interconnection must be provided and having confirmed that the problem is well posed, the implementation of a CNN in a pipelined fashion on an array of computational elements that implements a certain communication fabric is possible if, assigned its convolutional layers to the computational units, there exist communication channels that match the connectivity of the network. We reformulate the design of the topology by relating it in a bipartite fashion, on one side as high-level mapping, in terms of the graph representation of both CNNs and communication fabric topology, and on the other at the physical level, dealing with the feasibility of its implementation and targeting the specificities of computational memory. With regard to the former, given a communication fabric that implements a graph topology F and the directed graph representation C of a CNN, with vertices representing convolutional layers and edges representing activations directed toward the direction of computation, the CNN is executable in a pipelined fashion on a communication fabric implementing F if there exists a homomorphism $h:C\rightarrow F$. With regard to the implementation of the communication fabric, because of the non-negligible physical size of computational memory arrays with respect to their digital counterparts, the communication infrastructure also needs to exploit efficiently proximity in space of the computational units in order to meet the latency

```python
# Define variables and placeholders
weights1 = tf.get_variable("weights_1", [784, 250])
weights2 = tf.get_variable("weights_2", [250, 10])
input = tf.placeholder(shape=(None, 784))
label = tf.placeholder(shape=(None, 10))
# Activations
a1 = tf.sigmoid(tf.matmul(weights1, input))
a2 = tf.sigmoid(tf.matmul(weights2, a1))
# Loss function
loss = tf.nn.l2_loss(a2 - label)

# Train operation
optimizer =
tf.GradientDescentOptimizer(learning_rate=0.1)
train_op = optimizer.minimize(loss)
sess = tf.Session()
samples = ...
labels = ...

# For training:
sess.run(train_op, feed_dict={'input': samples,
        'label': labels})
```

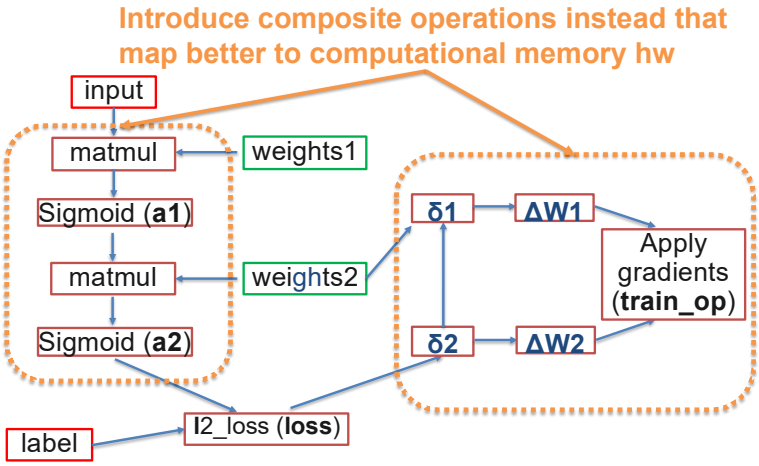

Figure 10 Training example in python pseudo-code of MNIST using the in-memory computing accelerator software stack on top of the Tensorflow framework.

requirements and allow attainable design of the communication circuits.

We have proposed in [54] a network topology that by construction is homomorphic to all state-of-art convolutional neural networks, while providing on the physical layer a scalable, efficient implementation. Examples of the mapping for feedforward and Inception-style connectivity on our proposed topology are displayed in **Figure 9**. The proposed topology is built on a unit graph equal to complete graph K_6, that is a graph with 6 vertices all connected with one another. The overall N-vertices graph topology is built by applying selected vector identification operations to a number of unit graphs. By construction, this topology is homomorphic to all four types of connectivity described above.

Delving now into the physical layer, the corresponding physical implementation of the unit graph sees I/O links implementing the complete connection of a 2-by-3 neighbourhood, with the bandwidth of the links tailored layers with the maximum number of channels. During execution, at each cycle the activations from one convolution are transferred by the links from one crossbar to another that implements the subsequent layer; the bandwidth of the links being tailored on the layers with the largest number of activations.

By enforcing the hardware connectivity for seamless pipelined execution, the aim moves at maximally exploiting the resources available in order to guarantee efficient area utilization of the crossbars and optimizing throughput and latency. Throughout the network, different kernels result at different area utilization of the crossbar. In particular, area utilization is quite poor for the first shallow layers. We can increase the crossbar areal efficiency by computing several activations of the same layer in parallel on the same crossbar, in a number proportional to the depth of each layer such that the bandwidth of the links is always fully exploited. While this method increases crossbar areal efficiency, it simultaneously also increases the throughput and decreases the latency of the CNNs implemented.

Lastly, this approach is also scalable to multichip implementations, by extending the on-chip connectivity of unit graphs from one chip to another with off-chip links.

5.2 Software stack

The software stack enables programmers to use the DNN accelerator without having to deal with underlying hardware complexities such as the interconnect topology, the characteristics of the chips, or the programming of the crossbar arrays. **Figure 10** illustrates an example of a user application in python pseudo-code that trains a simple two-layer neural network using the Tensorflow framework, with the respective dataflow graph shown on the right. The in-memory computing accelerator software stack provides plugins to existing frameworks, like Tensorflow, and is able to identify a set of training operators that can be fused into composite operators that map nicely to the in-memory computing accelerator hardware in a pipeline fashion. The software stack is able to compile the training dataflow graph into operators suitable for the accelerator, map the training pipeline to the hardware, and orchestrate data movement and synchronizations steps to and from the accelerator.

An example of an inference application in python pseudo-code for a DNN model based on the Resnet32 architecture using Imagenet [55] evaluation data, is shown in **Figure 11**. The example highlights the minimum set of user interfaces the software stack needs to support in order to perform inference: loading a dataset to be evaluated (load_dataset()), loading a pre-trained model (load_model()), executing the inference (eval), and retrieving prediction statistics (e.g., average_precision_call()). Support for loading a dataset can be implemented by re-using existing software libraries for the target dataset format (e.g., pandas, etc.). The main challenge of

the software stack is to understand and compile a supplied DNN model graph, map it efficiently to the hardware accelerator architecture, and orchestrate data movement to and from the accelerator. This complexity is completely hidden from the user and is mostly implemented within the load_model() API, with the inference execution and data movement taking place during the eval() API call.

We now discuss the DNN model graph compilation and its execution to the in-memory computing accelerator accelerator. To support the above high-level programming interface, the software stack is split into multiple sub-systems, including a compiler, a run-time library, and a driver.

The driver, a part of the operating system, is the lowest software layer and is responsible for enabling applications to program and use the accelerator. First, it is responsible for identifying the accelerator and providing applications with information about its capabilities. This information includes: the number of the analog cores as well as their size, their interconnection, the capabilities of the digital units, etc. The driver also exposes a low-level interface that allows applications to configure the accelerator. Specifically, using the driver applications can program the analog cores with specific synaptic weights, program the digital units, configure how data flows between the analog cores and digital units, as well as control how data are transferred from host memory to accelerator

```
import cmDNN as cm

DATASET_LOC = '/path/to/imagenet1k/test'
MODEL_LOC = '/path/to/onnx/resnet-32.model'

# Load dataset
X_eval,y_eval = cm.load_dataset(DATASET_LOC)

# Load model
resnet32 = cm.load_model(MODEL_LOC)

# Run inference on validation dataset
pred = resnet32.eval(X_eval, y_eval)

# Print accuracy stats for inference
print(cm.average_precision_call(pred))
```

Figure 11 Inference example in python pseudo-code of Resnet32 using the in-memory computing accelerator software stack.

memory and vice versa using direct-memory access (DMA) operations.

The run-time library provides a convenient, yet low-level, programming interface for applications to use the driver. It abstracts away details such as how the user-space program and the driver communicate, as well as implementing functionality commonly used by applications such as data structures for describing the configuration of the chip, as well as its capabilities.

Using the run-time library, a programmer can make full use of the accelerator. This is, however, a complicated endeavor and requires significant knowledge and expertise on behalf of the programmer about the inner workings of the accelerator, especially for achieving good performance. This is the equivalent of programming a CPU using machine language, and even though we expect some users to do so, we want to allow a wider class of applications to take advantage of the accelerator without requiring their programmers to fully program it.

To this end, the software stack also includes a complier. The compiler is responsible for accepting machine learning models (e.g., ONNX), which can be used both for training and inference, as input and mapping them for execution to the accelerator. This includes mapping the nodes of the machine learning model into accelerator functions, programming the crossbar arrays with the proper synaptic ways, and configuring the communication between the different accelerator units but also the host. The output of the compiler is a program that configures the chip using calls to the library and the driver, and then executes the given machine learning model by using the accelerator. The compiler is responsible for transforming operations (e.g., convolutions) in a proper form so that they can be executed by the accelerator. Because not all operations can be mapped into the accelerator, the compiler might choose to implement some computations on the host or even reject the machine learning model as incompatible.

5.3 Performance Assessment

In the previous subsections, we discussed how the computational memory can efficiently be used to map DNNs on-chip and perform forward inference and training. In this subsection, we will assess the energy efficiency and acceleration of the mixed-precision computational memory architecture for training.

We estimated the energy efficiency of the mixed-precision training architecture with respect to a corresponding 32-bit design based on the example two-layer perceptron used to perform MNIST digit classification. In both designs, all the digital memory needed for training was assumed to be implemented with on-chip static random-access memory (SRAM). The computational-memory-based mixed-precision design resulted in an average energy reduction of approximately 270x for the forward and backward propagation stages. Since we can substantially improve the energy efficiency of the data propagation stages, any optimization of the remaining digital update unit will result in significantly improved overall performance. For example, in our comparative study, the digital update unit in the mixed-precision training architecture was optimized. In particular, the outer-product based weight update digital computation was performed with reduced precision. The additional device programming overhead was negligible, since on average only one PCM device is programmed every two training images. All of this resulted in approximately 140x reduction in energy consumption for the weight update stage in mixed-precision in memory computing architecture with respect to the 32-bit design. Overall, the mixed-precision design could accelerate the training of the two-layer perceptron by approximately 10x and reduce the energy consumption by about 170x with respect to the 32-bit implementation.

Deep learning based on a mixed-precision computational memory architecture is expected to bring efficiency gains to larger and more complex networks as well. Using resistive memory arrays to store DNN weights allows large networks to be fit on-chip. Moreover, each array can perform the matrix-vector multiplications of each layer independently with very high efficiency and minimal intermediate data movements. In contrast to the conventional fully digital implementations, where the same resources are shared for all the computations, the dedicated computational memory arrays can enable efficient pipelining of data propagations across layers, which could be highly beneficial for very deep networks. On the other hand, compared to fully analog deep learning accelerators [31, 56], we require an additional high-precision memory to accumulate weight updates. However, the digital implementation of the training optimizer has several advantages. The digital optimizer compensates for the analog device non-idealities, extends its endurance and makes the trade-off between precision and accuracy more favorable. This also gives the computational memory architecture the flexibility to be configured to suit a wider class of deep learning models and algorithms.

For inference-only applications, where data is propagated through the network on offline-trained weights, the efficient matrix-vector multiplication realized via in-memory computing is very attractive. The energy efficiency of PCM-based computational memory for inference is expected to be comparable to that achieved using other non-volatile memory technologies such as RRAM. The state-of-the-art experimental demonstrations of DNN inference based on in-memory computing have reported competitive energy efficiencies between 10 and 100 TOPS/W for reduced-precision matrix-vector multiplications [57, 58]. However, the full system efficiency including array-to-array communication may be lower, depending on how efficiently the communication links can transmit data.

Conclusion

Due to the explosive growth of data-centric AI workloads and the imminent end of CMOS scaling laws, there is a significant need to explore alternate non-von Neumann computing architectures as well as post-silicon devices. In-memory computing is one such computing paradigm where certain computational tasks are performed in place in the memory by exploiting the physical attributes of memory devices such as memristive devices. A promising application domain for in-memory computing is deep learning and in this article we focused on a mixed-precision in-memory computing approach to training deep neural networks. The essential idea is to store the synaptic weights in a computational memory unit comprising crossbar arrays of memristive devices. During training, the matrix-vector multiply operations associated with the forward and backward passes are performed in place without the need to move around the synaptic weights. An additional conventional memory unit is used to store the accumulated weight updates in high precision and once they reach a threshold value, programming pulses are applied to the memristive devices to update the synaptic weights in place. This concept was validated using training simulations based on statistically accurate PCM models capturing the behavior of devices fabricated in 90 nm CMOS technology. A two-layer perceptron was trained to perform handwritten digit classification. A test accuracy of 98.31% was achieved which is only 0.11% lower than the equivalent classification accuracy achieved in high precision training. Subsequently, we presented several approaches to improve the computational capabilities of memristive devices. Multi-memristive synaptic architectures were shown to improve the conductance range as well as the stochasticity associated with the accumulative behavior. The concept of projected memory was presented that could mitigate the challenges associated with temporal variations in the conductance values. It was shown that it is possible to achieve remarkably high precision in-memory scalar multiplication (equivalent to 8-bit fixed point arithmetic) using projected PCM devices. Finally, we presented an overview of the system integration and software aspects of a DNN accelerator. A key challenge for the DNN accelerator is to provide a communication fabric that can efficiently move activations from one computational memory unit to another. A network topology built on a unit graph equal to complete graph K_6 was presented that by construction is homomorphic to all state-of-the-art convolutional neural networks. The software stack comprises three essential components namely, the OS driver, the compiler and a library that allows user applications to directly perform inference or training.

There is a significant effort towards the design of custom ASICs based on reduced precision arithmetic and highly optimized dataflow. However, one of the primary reasons for the inefficiency, namely the need to shuttle millions of synaptic weight values between the memory and processing units, remains unaddressed. The in-memory computing approach presented in this article addresses this key challenge. In summary, we believe that we will see two stages of innovations that take us from the near term, where the DL accelerators are built with conventional CMOS, towards a period of innovation involving the mixed-precision in-memory computing approach presented in this article.

Acknowledgment

We would like to thank our colleagues at IBM T. J. Watson Research Center and IBM Research – Almaden for their contributions to this work. We would also like to thank our academic collaborators from New Jersey Institute of Technology, University of Patras, EPFL and ETH Zurich.

References

1. P. A. Merolla, J. V. Arthur, R. Alvarez-Icaza, *et al.*, "A million spiking-neuron integrated circuit with a scalable communication network and interface," Science, vol. 345, no. 6197, pp. 668-673, 2014.

2. A. Sebastian, M. Le Gallo, G. W. Burr, S. Kim, M. BrightSky and E. Eleftheriou, "Tutorial: Brain-inspired computing using phase-change memory devices," Journal of Applied Physics, vol. 124, no. 111101, 2018.

3. G. W. Burr, M. J. Brightsky, A. Sebastian, *et al.*, "Recent Progress in Phase-Change Memory Technology," IEEE Journal on Emerging and Selected Topics in Circuits and Systems, vol. 6, no. 2, pp. 146-162, 2016.

4. J. Borghetti, G. S. Snider, P. J. Kuekes, J. J. Yang, D. R. Stewart, and R. S. Williams, "'Memristive' switches enable 'stateful' logic operations via material implication," Nature, vol. 464, p. 873–876, 2010.

5. J. J. Yang, D. B. Strukov and D. R. Stewart, "Memristive devices for computing," Nature Nanotechnology, vol. 8, pp. 13-24, 2013.

6. M. Di Ventra and Y. V. Pershin, "The parallel approach," Nature Physics, vol. 9, pp. 200-202, 2013.

7. M. A. Zidan, J. P. Strachan, and W. D. Lu, "The future of electronics based on memristive systems," Nature Electronics, vol. 1, pp. 22-29, 2018.

8. D. Ielmini and H.-S. P. Wong, "In-memory computing with resistive switching devices," Nature Electronics, vol. 1, p. 333–343, 2018.

9. A. Biswas and A. P. Chandrakasan, "Conv-RAM: An energy-efficient SRAM with embedded convolution computation for low-power CNN-based machine learning applications," IEEE International Solid - State Circuits Conference - (ISSCC), pp. 488-490, 2018.

10. M. Cassinerio, N. Ciocchini and D. Ielmini, "Logic Computation in Phase Change Materials by Threshold and Memory Switching," Advanced Materials, vol. 25, no. 5975, 2013.

11. V. Seshadri, D. Lee, T. Mullins, H. Hassan, A. Boroumand, J. Kim, M. A. Kozuch, O. Mutlu, P. B. Gibbons and T. C. Mowry, "Buddy-RAM: Improving the Performance and Efficiency of Bulk Bitwise Operations Using DRAM," arXiv preprint, arXiv:1611.09988, 2016.

12. C. D. Wright, P. Hosseini and J. A. V. Diosdado, "Beyond von-Neumann Computing with Nanoscale Phase-Change Memory Devices," Advanced Functional Materials, vol. 23, no. 2248, 2013.

13. P. Hosseini, A. Sebastian, N. Papandreou, C. D. Wright and H. Bhaskaran, "Accumulation-Based Computing Using Phase-Change Memories With FET Access Devices," IEEE Electron Device Letters, vol. 36, no. 9, 2015.

14. A. Sebastian, T. Tuma, N. Papandreou, M. Le Gallo, L. Kull, T. Parnell and E. Eleftheriou, "Temporal correlation detection using computational phase-change memory," Nature Communications, vol. 8, no. 1115, 2017.

15. M. Le Gallo, A. Sebastian, G. Cherubini, H. Giefers and E. Eleftheriou, "Compressed sensing recovery using computational memory," IEEE International Electron Devices Meeting (IEDM), vol. 65, no. 10, pp. 28.3.1 - 28.3.42, 2017.

16. M. Le Gallo, A. Sebastian, G. Cherubini, H. Giefers and E. Eleftheriou, "Compressed Sensing With Approximate Message Passing Using In-Memory Computing," IEEE Transactions on Electron Devices, pp. 4304 - 4312, 2018.

17. M. Hu, C. E. Graves, C. Li, *et al.*, "Memristor-based analog computation and neural network classification with a dot product engine," Advanced Materials, vol. 30, no. 9, p. 1705914, 2017.

18. F. Merrikh-Bayat, X. Guo, M. Klachko, M. Prezioso, K. K. Likharev and D. B. Strukov, "High-Performance Mixed-Signal Neurocomputing With Nanoscale Floating-Gate Memory Cell Arrays," IEEE Transactions on Neural Networks and Learning Systems, vol. 29, no. 10, pp. 4782 - 4790, 2018.

19. Z. Sun, G. Pedretti, E. Ambrosi, A. Bricalli, W. Wang and D. Ielmini, "Solving matrix equations in one step with cross-point resistive arrays," Proceedings of the National Academy of Sciences of the United States of America (PNAS), vol. 10, no. 116, pp. 4123-4128, 2019.

20. M. Le Gallo, A. Sebastian, R. Mathis, *et al.*, "Mixed-Precision In-Memory Computing," Nature Electronics, vol. 1, pp. 246-253, 2018.

21. S. R. Nandakumar, M. Le Gallo, I. Boybat, B. Rajendran, A. Sebastian and E. Eleftheriou, "Mixed-precision architecture based on computational memory for training deep neural networks," IEEE International Symposium on Circuits and Systems (ISCAS), pp. 1 - 5, 2018.

22. S. R. Nandakumar, M. Le Gallo, I. Boybat, B. Rajendran, A. Sebastian, and E. Eleftheriou , "Mixed-precision training of deep neural networks using computational memory," arXiv preprint, arXiv:1712.01192, 2017.

23. A. Sebastian, I. Boybat, M. Dazzi, *et al*., "Computational memory-based inference and training of deep neural networks," 2019 Symposium on VLSI Technology, Kyoto, Japan, 2019, pp. T168-T169.

24. S. Gupta, A. Agrawal, K. Gopalakrishnan and P. Narayanan, "Deep learning with limited numerical precision," 32nd International Conference on Machine Learning (ICML), vol. 37, pp. 1737-1746, 2015.

25. L. K. Muller and G. Indiveri, "Rounding Methods for Neural Networks with Low Resolution Synaptic Weights," arXiv preprint, arXiv:1504.05767, 2015.

26. M. Courbariaux, Y. Bengio, and J.-P. David, "BinaryConnect: Training Deep Neural Networks with binary weights during propagations," Advances in Neural Information Processing Systems (NIPS), pp. 3123-3131, 2015.

27. P. Merolla, R. Appuswamy, J. Arthur, S. K. Esser and D. Modha, "Deep neural networks are robust to weight binarization and other non-linear distortions," arXiv preprint, arXiv:1606.01981, 2016.

28. H. Zhang, J. Li, K. Kara, D. Alistarh, J. Liu and C. Zhang, "ZipML: Training Linear Models with End-to-End Low Precision, and a Little Bit of Deep Learning," 34th International Conference on Machine Learning (ICML), vol. 70, pp. 4035-4043, 2017.

29. T. Gokmen and Y. Vlasov, "Acceleration of Deep Neural Network Training with Resistive Cross-Point Devices: Design Considerations," Frontiers of Neuroscience, vol. 333, no. 10, 2016.

30. G. W. Burr *et al.*, "Experimental Demonstration and Tolerancing of a Large-Scale Neural Network (165 000 Synapses) Using Phase-Change Memory as the Synaptic Weight Element," IEEE Transactions on Electron Devices, vol. 62, no. 11, pp. 3498 - 3507, 2015.

31. S. Ambrogio, P. Narayanan, H. Tsai, *et al.*, "Equivalent-accuracy accelerated neural-network training using analogue memory," Nature, vol. 558, p. 60–67, 2018.

32. S. R. Nandakumar *et al.*, "Mixed-precision deep learning based on computational memory," unpublished.

33. T. Tuma, A. Pantazi, M. Le Gallo, A. Sebastian and E. Eleftheriou, "Stochastic phase-change neurons," Nature Nanotechnology, vol. 11, p. 693–699, 2016.

34. T. Tuma, M. Le Gallo, A. Sebastian and E. Eleftheriou, "Detecting Correlations Using Phase-Change Neurons and Synapses," IEEE Electron Device Letters, vol. 37, no. 9, pp. 1238 - 1241, 2016.

35. A. Sebastian, M. Le Gallo and D. Krebs, "Crystal growth within a phase change memory cell," Nature Communications, vol. 5, no. 4314, 2014.

36. M. Le Gallo, T. Tuma, F. Zipoli, A. Sebastian and E. Eleftheriou, "Inherent stochasticity in phase-change memory devices," 46th European Solid-State Device Research Conference (ESSDERC), pp. 373-376, 2016.

37. S. R. Nandakumar, M. Le Gallo, I. Boybat, B. Rajendran, A. Sebastian and E. Eleftheriou, "A phase-change memory model for neuromorphic computing," Journal of Applied Physics, vol. 124, no. 152135 , 2018.

38. IBM, 2018. [Online]. Available: https://analog-ai-demo.mybluemix.net.

39. S. R. Nandakumar, I. Boybat, V. Joshi, C. Piveteau, M. Le Gallo, B. Rajendran, A. Sebastian, and E. Eleftheriou, "Phase-change memory models for deep learning training and inference," in IEEE International Conference on Electronics, Circuits and Systems (ICECS) (2019), submitted for publication.

40. I. Boybat, M. Le Gallo, T. Moraitis, Y. Leblebici, A. Sebastian and E. Eleftheriou, "Stochastic weight updates in phase-change memory-based synapses and their influence on artificial neural networks," 13th Conference on Ph.D. Research in Microelectronics and Electronics (PRIME), pp. 13 - 16, 2017.

41. I. Boybat, M. L. Gallo, S. R. Nandakumar, T. Moraitis, T. Parnell, T. Tuma, B. Rajendran, Y. Leblebici, A. Sebastian and E. Eleftheriou, "Neuromorphic computing with multi-memristive synapses," Nature Communications, vol. 9, no. 2514, 2018.

42. I. Boybat, S. R. Nandakumar, M. L. Gallo, B. Rajendran, Y. Leblebici, A. Sebastian and E. Eleftheriou, "Impact of conductance drift on multi-PCM synaptic architectures," 2018 Non-Volatile Memory Technology Symposium (NVMTS), pp. 1-4, 2018.

43. I. Boybat, C. Giovinazzo, E. Shahrabi, *et al*., "Multi-ReRAM Synapses for Artificial Neural Network Training," IEEE International Symposium on Circuits and Systems (ISCAS), pp. 1-5, 2019.

44. S. Kim, N. Sosa, M. BrightSky, D. Mori, W. Kim, Y. Zhu, K. Suu and C. Lam, "A phase change memory cell with metallic surfactant layer as a resistance drift stabilizer," IEEE International Electron Devices Meeting (IEDM), pp. 30.7.1 - 30.7.4 , 2013.

45. W. W. Koelmans, A. Sebastian, V. P. Jonnalagadda, D. Krebs, L. Dellmann, and E. Eleftheriou, "Projected phase-change memory devices," Nature Communications, vol. 6, no. 8181, 2015.

46. I. Giannopoulos, A. Sebastian, M. Le Gallo, V. P. Jonnalagadda, M. Sousa, M. N. Boon, and E. Eleftheriou, "8-bit Precision In-Memory Multiplication with Projected Phase-Change Memory," EEE International Electron Devices Meeting (IEDM), pp. 27.7.1-27..7.4, 2018.

47. M. Le Gallo, D. Krebs, F. Zipoli, M. Salinga, and A. Sebastian, "Collective Structural Relaxation in Phase-Change Memory Devices," Advanced Electronic Materials, vol. 4, no. 9, 2018.

48. M. Abadi, P. Barham, J. Chen et al., "TensorFlow: A System for Large-Scale Machine Learning," 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI), p. 265–283, 2016.

49. "ONNX. Open Neural Network Exchange," 2017. [Online]. Available: https://github.com/onnx/onnx.

50. Y. Lecun, L. Bottou, Y. Bengio and P. Haffner, "Gradient-based learning applied to document recognition," Proceedings of the IEEE , vol. 86, no. 11, pp. 2278 - 2324, 1998.

51. C. Szegedy, W. Liu, Y. Jia, *et al.*, "Going deeper with convolutions," IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 1-9, 2015.

52. K. He, X. Zhang, S. Ren and J. Sun, "Deep residual learning for image recognition," in IEEE conference on computer vision and pattern recognition (CVPR), 2016.

53. G. Huang, Z. Liu, L. van der Maaten, *et al.*, "Densely Connected Convolutional Networks," IEEE conference on computer vision and pattern recognition, pp. 2736-2744, 2017.

54. M. Dazzi, A. Sebastian, P. A. Francese, *et al.*, "5 Parallel Prism: a topology for communication-efficient implementations of convolutional neural networks on computational memory," arXiv preprint, arXiv:1906.03474, 2019.

55. J. Deng, W. Dong, R. Socher, L. Li, K. Li and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2009.

56. S. Kim, T. Gokmen, H. Lee and W. E. Haensch, "Analog CMOS-based resistive processing unit for deep neural network training," in IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS), 2017.

57. C. Xue, W. Chen, J. Liu, J. Li, W. Lin, W. Lin, J. Wang, W. Wei, T. Chang, T. Chang, T. Huang, H. Kao, S. Wei, Y. Chiu, C. Lee, C. Lo, Y. King, C. Lin, R. Liu, C. Hsieh, K. Tang, and M. Chang, "A 1Mb Multibit ReRAM Computing-In-Memory Macro with 14.6ns Parallel MAC Computing Time for CNN-Based AI Edge Processors, in IEEE International Solid- State Circuits Conference (ISSCC), pp. 388–390, 2019.

58. M. Hu, C. E. Graves, C. Li, *et al.*, Memristor-based analog computation and neural network classification with a dot product engine, Advanced Materials, vol. 30, pp. 1705914, 2018.

E. Eleftheriou *IBM Research – Zurich, 8803 Rüschlikon, Switzerland (ele@zurich.ibm.com).* Author's biography appears here.

M. Le Gallo *IBM Research – Zurich, 8803 Rüschlikon, Switzerland (anu@zurich.ibm.com).* Author's biography appears here.

S.R. Nandakumar *IBM Research – Zurich, 8803 Rüschlikon, Switzerland (nsi@zurich.ibm.com).* Author's biography appears here.

C. Piveteau *IBM Research – Zurich, 8803 Rüschlikon, Switzerland (piv@zurich.ibm.com).* Author's biography appears here.

I. Boybat *IBM Research – Zurich, 8803 Rüschlikon, Switzerland (ibo@zurich.ibm.com).* Author's biography appears here.

V. Joshi *IBM Research – Zurich, 8803 Rüschlikon, Switzerland (vin@zurich.ibm.com).* Author's biography appears here.

R. Khaddam-Aljameh *IBM Research – Zurich, 8803 Rüschlikon, Switzerland (rid@zurich.ibm.com).* Author's biography appears here.

M. Dazzi *IBM Research – Zurich, 8803 Rüschlikon, Switzerland (daz@zurich.ibm.com).* Author's biography appears here.

I. Giannopoulos *IBM Research – Zurich, 8803 Rüschlikon, Switzerland (nno@zurich.ibm.com).* Author's biography appears here.

G. Karunaratne *IBM Research – Zurich, 8803 Rüschlikon, Switzerland (kar@zurich.ibm.com).* Author's biography appears here.

B. Kersting *IBM Research – Zurich, 8803 Rüschlikon, Switzerland (bke@zurich.ibm.com).* Author's biography appears here.

M. Stanisavljevic *IBM Research – Zurich, 8803 Rüschlikon, Switzerland (ysm@zurich.ibm.com).* Author's biography appears here.

V. P. Jonnalagadda *IBM Research – Zurich, 8803 Rüschlikon, Switzerland (vjo@zurich.ibm.com).* Author's biography appears here.

N. Ioannou *IBM Research – Zurich, 8803 Rüschlikon, Switzerland (nio@zurich.ibm.com).* Author's biography appears here.

K. Kourtis *IBM Research – Zurich, 8803 Rüschlikon, Switzerland (kou@zurich.ibm.com).* Author's biography appears here.

P. A. Francese *IBM Research – Zurich, 8803 Rüschlikon, Switzerland (pfr@zurich.ibm.com).* Author's biography appears here.

A. Sebastian *IBM Research – Zurich, 8803 Rüschlikon, Switzerland (ase@zurich.ibm.com).* Author's biography appears here.