Toward on-chip acceleration of the backpropagation algorithm using nonvolatile memory

P. Narayanan
A. Fumarola
L. L. Sanches
K. Hosokawa
S. C. Lewis
R. M. Shelby
G. W. Burr

By performing computation at the location of data, non-Von Neumann (VN) computing should provide power and speed benefits over conventional (e.g., VN-based) approaches to data-centric workloads such as deep learning. For the on-chip training of large-scale deep neural networks using nonvolatile memory (NVM) based synapses, success will require performance levels (e.g., deep neural network classification accuracies) that are competitive with conventional approaches despite the inherent imperfections of such NVM devices, and will also require massively parallel yet low-power read and write access. In this paper, we focus on the latter requirement, and outline the engineering tradeoffs in performing parallel reads and writes to large arrays of NVM devices to implement this acceleration through what is, at least locally, analog computing. We address how the circuit requirements for this new neuromorphic computing approach are somewhat reminiscent of, yet significantly different from, the well-known requirements found in conventional memory applications. We discuss tradeoffs that can influence both the effective acceleration factor ("speed") and power requirements of such on-chip learning accelerators.

Introduction

The extreme flexibility of digital circuits has allowed modern processors based on the Von Neumann architecture to not only efficiently implement algorithms for a wide variety of problems, but to consistently improve system performance at an exponential rate. However, with continued device scaling constrained by power and voltage considerations, the time and energy spent transporting data between memory and processor (across the so-called "Von Neumann bottleneck") has become problematic for data-centric applications such as real-time image recognition and natural language processing.

One example of non–Von Neumann computing is the human brain. Characterized by its massively parallel architecture and adaptive elements (e.g., its synapses), the brain can outperform modern processors on many tasks involving unstructured data classification and pattern recognition. Artificial neural networks (ANNs), first conceived in the mid-1940s to mimic what was then known about neural systems, perform computations in a naturally parallel fashion. Modern graphical processing units (GPUs) have greatly increased both the size of the networks and the datasets that can be trained in reasonable time. In turn, this has commensurately improved classification performance to the point that these systems are now becoming commercially pervasive. In contrast to power-hungry GPUs, IBM's TrueNorth chip is a flexible and modular non-VN tool for implementing forward inference of large ANNs [e.g., deep neural networks (DNNs)] at ultralow power [1]. Synaptic weights are typically trained off-line and transferred onto digital SRAM (static random access memory) arrays to perform forward propagation of large and complex DNNs.

One path for extending such non-VN systems toward full on-chip learning—and thus to provide *accelerated DNN training* at lower power than GPUs—is to replace these reliable but binary SRAM memory cells used in TrueNorth with dense and analog (but less reliable) nonvolatile memory (NVM). By performing computation at the

Digital Object Identifier: 10.1147/JRD.2017.2716579

location of data, such an approach—on-chip training of large-scale DNN using NVM-based synapses [2–8]—could potentially provide significant power and speed benefits on the specific and important task of training very large DNNs. Such an implementation can realize the multiply-accumulate (MAC) operations at the heart of most neural network algorithms extremely efficiently, using physics—Ohm's law followed by current summation (Kirchhoff's current law)—for locally analog computation at the location of the weight data. The idea of performing MAC operations in the analog domain for neuromorphic computing is quite old [9, 10], but recent developments in NVM devices offer new opportunities for revisiting this idea [2, 11, 12].

However, practical viability of such an approach has several requirements. First, despite the inherent imperfections of NVM devices such as phase change memory (PCM) [2, 3] or resistive random access memory (RRAM) [5], such NVM-based networks *must* achieve similar performance levels (e.g., classification accuracies) when compared to DNNs trained using CPUs (central processing units) or GPUs. Second, the benefits of performing computation at the data must confer a decided advantage in either training power or speed (or preferably, both) [4]. Finally, any on-chip accelerator should be applicable toward networks of different types (e.g. fully-connected multi-layer perceptions and convolutional neural networks) and be reconfigurable for networks of different shapes (wide, with many neurons, or deep, with many layers) [4].

In this paper, we outline the engineering tradeoffs in performing parallel reads and writes to large arrays of NVM devices to implement this acceleration through what is, at least locally analog computing. In contrast to others who have treated this as a digital design problem fed by conventional [13] or emerging NVM devices [14, 15], or who have focused solely on the NVM crossbar array without much focus on peripheral circuitry [16–21], we are trying to consider this problem as a holistic, mixed analog-digital-NVM design problem. We do not address here spiking neural networks, in which synaptic plasticity is modified based on the timing of sparse upstream and downstream neuronal spikes [22–24], since such local learning rules have not yet been harnessed by a global learning algorithm exhibiting the kind of convergence properties (and thus demonstrated application success) offered by backpropagation.

In this paper, we attempt to lay out the challenges for the design of an NVM-based on-chip accelerator for backpropagation training. We plan to publish a full power and speed analysis in the future, once we can fully disclose the specific design choices we have identified to address these challenges. In this paper, we address the roles that peripheral circuitry must fulfill within such a system, and discuss how the resulting circuit requirements for this new neuromorphic computing approach are different from the well-known requirements found in conventional memory applications. We discuss tradeoffs that can influence both the effective acceleration factor (speed) and power requirements of such on-chip learning accelerators.

Recap of previous work

In this section, we briefly review our previous work [2–8] in assessing the potential accuracy, speed, and power of on-chip NVM-based DNN training. **Figure 1** shows our general concept for fully connected DNN layers. In our approach, each synaptic layer in a large ANN—connecting an upstream layer of neurons (either input or hidden neurons) to the next downstream layer of neurons (either hidden or output neurons)—is mapped to one or more "crossbar arrays" of NVM device-pairs. Each synapse is composed of two conductances, encoding synaptic weight in the difference between these two conductances, $w_{ij} = G^{+}_{ij} - G^{-}_{ij}$, with *i* and *j* being indices within the 2D array of synapses. In this way, negative and positive weights can be readily encoded using positive-only conductance values. Each crosspoint, or intersection between a horizontal "row-line" and vertical "column-line," also contains an access device, either a 2-terminal selector [25] or a 3-terminal transistor as shown in Figure 1. This access device ensures that external circuitry can precisely target which device or devices will participate in read and write operations. The requirements and specifications of a 2-terminal selector, and of the voltage selection scheme, for this kind of neuromorphic application remain mostly unchanged from the requirements for a conventional memory application [21, 25]: leakage must be minimized so that write power is delivered efficiently and reads performed accurately.

Device-related previous work

Using two phase-change memory devices per synapse, we demonstrated a three-layer perceptron (fully connected DNN) with 164,885 synapses [2], trained with the backpropagation algorithm [26] on a subset (5,000 examples) of the MNIST (modified National Institute of Standards and Technology) database of handwritten digits [27], using a modified weight-update rule, compatible with NVM and access device crossbar arrays [2]. We proved that this weight-update modification does not degrade the high "test" (generalization) accuracies such a three-layer network inherently delivers with respect to this problem when trained in software [2]. However, nonlinearity and asymmetry in PCM conductance response limited both "training" and "test" accuracy in our original, mixed hardware-software experiments to 82% to 83% [2]. Since our original paper was published, other researchers have proposed and verified similar update strategies, and studied the impact of device imperfections [16, 19, 20, 28].

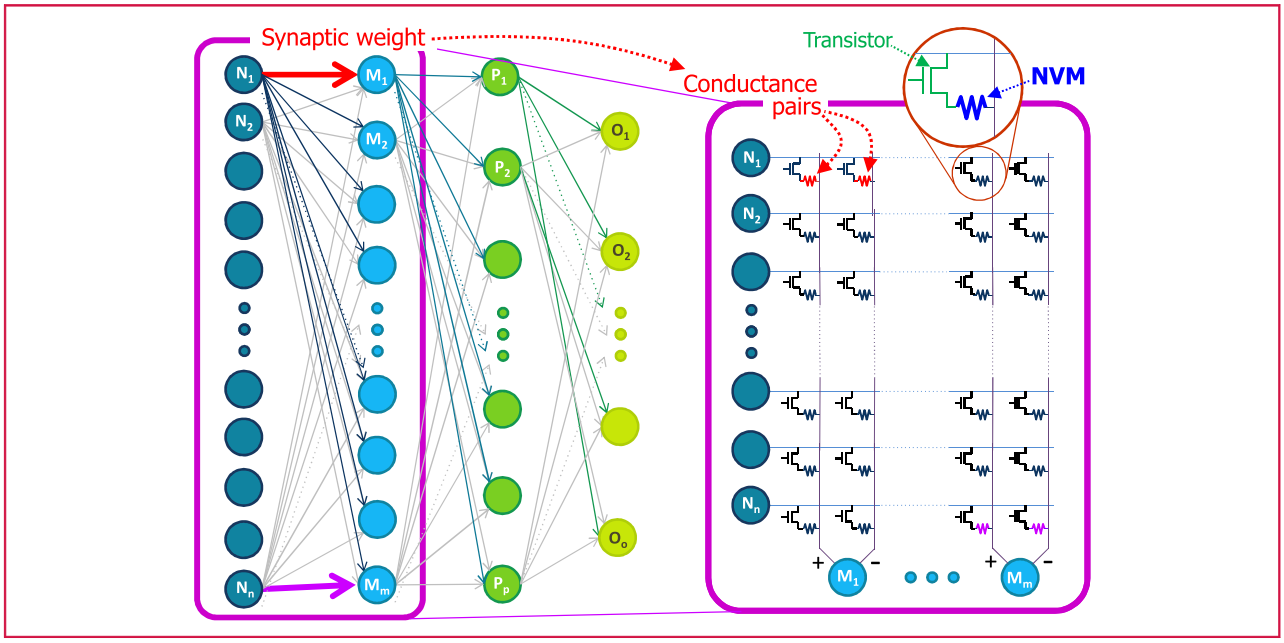

Figure 1

Neuro-inspired non–Von Neumann computing [1, 2], in which neurons activate each other through dense networks of programmable synaptic weights, can be implemented using dense crossbar arrays of nonvolatile memory (NVM) and 2-terminal or 3-terminal selector device-pairs.

Asymmetry (between the gradual conductance increases of PCM partial-SET and the abruptness of PCM RESET) was mitigated by an "Occasional RESET" strategy, which could be both infrequent and inaccurate [2]. While in these initial experiments, network parameters such as learning rate, η, had to be tuned very carefully, a modified "Local Gains" algorithm offered wider tolerance to η, higher classification accuracies, and lower training energy [4].

Tolerancing results—where we varied individual device and system parameters to gauge the effect of each on DNN accuracy—showed that while NVM-based DNNs can be expected to be highly resilient to random effects (NVM variability, yield, and stochasticity), they will be highly sensitive to "gradient" effects that act to steer all synaptic weights in a particular direction [2]. We showed that a bidirectional NVM with a symmetric, linear conductance response of finite but large dynamic range (e.g., each conductance step is relatively small) can deliver the same high classification accuracies on the MNIST digits as a conventional, software-based implementation [3]. One key observation is the importance of avoiding constraints on weight magnitude that arise when the two conductances (G^+ and G^-) are either both small or both large—e.g., synapses should remain in the center stripe of the "G-diamond" [2, 3] formed by the finite range of these two conductance values.

In subsequent papers, we have extended these observations and addressed several different yet useful topics. We have assessed the impact of undesired, time-varying conductance change, including drift in PCM and leakage of analog CMOS capacitors [7]. We have introduced a "jump-table" concept to describe the full cumulative distribution function (CDF) of conductance-change at each device conductance value [2], allowing the practical modeling of both the potentiation (SET) and depression (RESET) characteristics of real NVM devices [2, 6–8]. Using artificially constructed "jump-tables," we have studied the impact on DNN accuracy of various imperfections exhibited by such real NVM devices [6]. Finally, we have investigated the use of non-filamentary, bidirectional RRAM devices based on PrCaMnO (PCMO), as a potential approach for developing material variants that provide suitably linear conductance change [5, 7].

Circuit-related previous work

We have also explored tradeoffs in designing peripheral circuitry, balancing simplicity and area-efficiency with respect to the impact on DNN performance [7, 8]. We performed these studies for both fully bidirectional NVMs (such as PCMO), as well as for devices with abrupt conductance change in one direction (such as PCM). For fully bidirectional devices, a combination of algorithm approximations—including a piece-wise linear (PWL) nonlinear squashing function, a simple step-function derivative, limitations on the number of neuron states, and careful selection of programming pulses—can enable

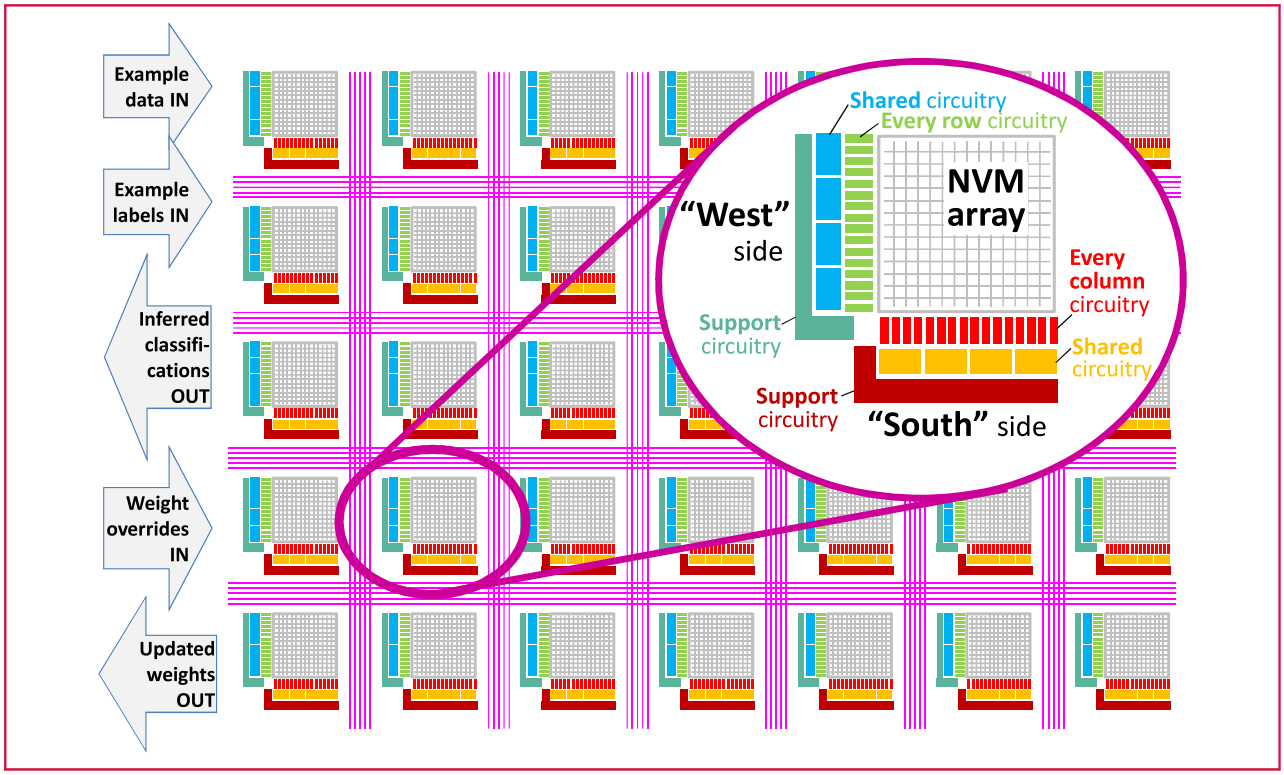

Figure 2

Proposed chip architecture for the acceleration of neural network training with analog non-volatile memory. A flexible routing network has three tasks: 1) to convey chip inputs (such as example data, example labels, and weight overrides) from the edge of the chip to the device arrays, 2) to carry chip outputs (such as inferred classifications and updated weights) from the arrays to the edge of the chip, and 3) to interconnect the various arrays in order to implement multi-layer neural networks. Each NVM array has input neurons (here shown on the "West" side of each array) and output neurons ("South" side), connected with a dense grid of synaptic connections. Peripheral circuitry is divided into circuitry assigned to individual rows and columns, circuitry shared between a number of neighboring rows and columns, and support circuitry.

competitive DNN performance [8]. We demonstrated that these approximations do not significantly degrade classification accuracies, compared to neuron implementations with rigorously precise functionality [8].

Devices with abrupt conductance change in one direction require an "Occasional RESET" operation, during which all conductances are measured. Synapses for which both G^+ and G^- are rather large then have both conductances decreased (by RESET followed by incremental SET) such that their difference, and thus the synaptic weight, remain the same. We showed that this measurement operation can be performed with coarse binning of weight values, but that real PCM exhibiting significant pulse-to-pulse variability may require read-verify steps during the incremental SET operations [8].

Finally, we have assessed the potential advantages, in terms of speed and power, of on-chip machine learning of large-scale DNN using NVM-based synapses, in comparison with conventional GPU-based hardware [4]. Under moderately aggressive assumptions for parallel-read and parallel-write speed (see [4] for details), PCM-based on-chip machine learning can potentially offer lower power and faster training (per DNN example) than GPU-based training for both large and small networks, even with the time and energy required for "Occasional RESET." Critical here is the design of area-efficient read/write circuitry, so that many copies of this circuitry operate in parallel (each handling a small number of columns (and rows), c_s).

In the remainder of this paper, we discuss the requirements for the peripheral circuitry that will be needed to support on-chip training of large-scale DNN using NVM-based synapses.

Generic architecture for on-chip learning

Figure 2 shows a schematic of a chip architecture for the acceleration of neural network training with analog NVM. Similar to the TrueNorth chip [1], the architecture is composed of a large number of identical array-blocks connected by a flexible routing network. Each array-block here represents a large NVM device array, perhaps of size

512 × 512 or 1,024 × 1,024 synaptic unit-cells. With a unit-cell composed of two NVM conductances, this would correspond to arrays of 512 × 1,024 or 1,024 × 2,048 NVM devices.

However, unlike TrueNorth [1], the routing network here is not a true point-to-point network, offering flexible connection from the "South side" of any single column to the "West side" of any single row. (We discuss connections to only one side of the array-block to simplify the discussion; use of all four sides for peripheral circuitry is certainly desirable for efficient use of chip area.) Instead, the routing network we envision need only be block-to-block, connecting all the columns at the "South side" to all the rows at the "West side" of the "next downstream" array-block. In some cases, connections might be needed between the "South side" of one array-block and the "South side" of another, mimicking an extra-tall array-block to implement neuron layers that have more neurons than the number of rows in any one array block. Similarly, routing between the "West side" of one array-block and the "West side" of another could be used to effectively implement very wide blocks, for DNNs that fan out from a small number of input neurons to hidden layers with a large number of neurons.

In addition to interconnecting the various array-blocks in order to implement multi-layer neural networks, this flexible routing network has two additional tasks. First, it must convey chip inputs such as example data (e.g., pixels of a training image), example labels (e.g., the ground truth classification of that training image), and weight overrides (see discussion toward the end of the paper) from the edge of the chip to the device arrays. Second, the routing network is needed to carry chip outputs—such as inferred classifications and updated weights—from the arrays to the edge of the chip. While chip inputs and outputs need to be directed to only a small subset of arrays on the chip (corresponding to the locations of input and output neurons), weight overrides and updated weights would potentially need to be sent to and received from every single array on-chip, making these global updates the far more challenging routing requirement. Furthermore, in order to enable scale-out to even larger DNNs with a multi-chip modular approach, the I/O interface protocol at the edge of the chip should be identical to the internal within-chip protocol.

Each NVM array-block has input neurons (here shown on the "West" side of each array) and output neurons ("South" side), connected with a dense grid of synaptic connections. Peripheral circuitry is divided into circuitry assigned to individual rows and columns, circuitry shared between a number of neighboring rows and columns, and support circuitry. In the actual mapping of DNN to array-blocks, each hidden layer neuron actually corresponds to two different pieces of circuitry: a particular instance of the each-column circuitry sitting at the "South side" of the array-block corresponding to an "output" from an upstream synaptic layer; and a matching instance of the each-row circuitry sitting at the "West side" of a different array-block, corresponding to an "input" to a downstream synaptic layer for the given neuron.

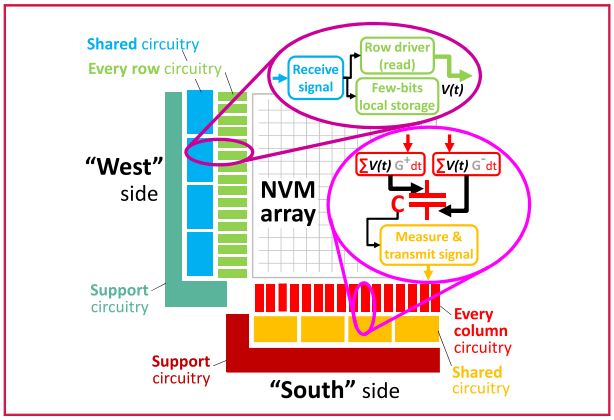

Figure 3

During forward propagation, signals arriving at the "West side" of each array-block are passed to row-drivers, generating a $V(t)$ signal on horizontal row-lines that is proportional to each upstream neuron excitation, x. (During this operation, a small number of bits encoding this x value—for later use during weight-update—are stored locally near the end of the row.) The resulting aggregate read currents from the entire column, e.g., $\sum V(t)G+$ and $\sum V(t)G-$, arrive at the peripheral circuitry located at the "South side" of each column, where they are integrated onto a single capacitor (either adding to, or subtracting from, its voltage as appropriate). After the integration is complete, each capacitor voltage must be measured and transmitted across the flexible routing network to its destination. This can either be the "South side" of another array-block (to effectively implement one extra-tall array block), or the "West side" of the "next downstream" array-block, where the transmitted signal continues the forward-propagation operation of the next synaptic layer.

In the next few sections, we describe the various tasks involved in DNN training, and how these tasks place unique demands and requirements on the peripheral circuitry of these array-blocks.

Forward propagation

Forward propagation or inference in a neural network involves the calculation of the neuron activations of a hidden/output layer, based on the neuron activations of the previous layer and the intervening synaptic weights. Here, we focus on fully connected neural networks, leaving the implementation of a convolutional neural network to a subsequent paper. **Figure 3** shows how the peripheral circuitry at the "West" and "South" sides of an array-block must work together to perform this operation. Signals arriving at the "West side" of the array-block are passed to row-drivers, generating a $V(t)$ (voltage as a function of time) signal on horizontal row-lines to convey a signal

proportional to the upstream neuron excitation, x. During this operation, a small number of bits encoding this x value must be stored locally near the end of the row. These bits will be needed later during weight-update.

In a conventional memory application, a read operation would activate only one row. Since a larger number of rows will be activated in this neuromorphic read operation (including, potentially, all of the rows), the circuit design point should allow for a sufficiently low current through each of the active crosspoints during forward read. The key constraints include the following: 1) The aggregate power and power density must be within manageable limits; 2) Aggregate current "in most cases" should be within the driving capability of peripheral driver circuits (it would be acceptable if, on a small number of examples/instances, the actual current ended up being less than what is expected from the aggregate sum, given the error tolerance of the algorithm); and 3) This current should stay within the long-term electromigration limits of the technology. Given that read voltages cannot be scaled arbitrarily due to noise, *IR* (current times resistance) drop, and variation considerations, it is of critical importance that the NVM element operate in a low but linear conductance regime to keep crosspoint currents (and thereby aggregate currents) within manageable limits.

The resulting aggregate multiply-accumulate currents from the entire column, e.g., $\sum V(t)G^{+}$ and $\sum V(\mathrm{t})G^{-}$, arrive at the peripheral circuitry located at the "South side" of each column, where they are integrated onto a single capacitor. Current mirrors can be used to add to, or subtract from, a pre-charged capacitor voltage as appropriate. Different column-lines that receive different total currents will have different voltages, but decoupled from the voltage to which any capacitor could be charged. Because of these slight differences in voltage, however, a fully passive scheme with no selector or transistor cannot guarantee a zero voltage difference across every "unselected" device. Since the currents here are the aggregate read current from a large number of rows, they tend to be significantly larger in magnitude than the memory read current due to only one NVM element. As such, sub-unity gain in the current mirrors can be used to control the mapping between this large current and a suitably small voltage change on the capacitor, thus avoiding the need for large and area-inefficient capacitors at every column.

The integration operation can be performed in multiple segments if needed—provided the scaled and mirrored currents are sufficiently small to prevent the capacitor voltage from saturating. After all the signals have been integrated, each capacitor voltage must be measured and transmitted across the flexible routing network to its destination. This can either be the "South side" of another array-block (to effectively implement an "extra-tall" array block), or the "West side" of the "next downstream" array-block, where the transmitted signal continues the forward-propagation operation of the next synaptic layer.

To ensure that the computation of a multilayer DNN does not readily collapse to a single linear equation and reduce its capabilities, a nonlinear squashing function is typically applied to the integrated signal at the periphery. Commonly used functions in software implementations include rectified linear units (ReLUs), tanh() (i.e., the hyperbolic-tangent function), or the logistic function ($S = 1/(1 + \exp(-t))$). However, the latter such functions are difficult to implement exactly unless a large number of transistors are included. (The ReLU is problematic because its unbounded nature is inconsistent with capacitors that cannot be charged beyond the fixed supply voltage.) Analog-to-digital and digital-to-analog converters tend to require a significant amount of chip real-estate, which is inconsistent with a small circuit-sharing parameter, c_s. As discussed earlier, a piece-wise linear (PWL) implementation of this squashing function is much more straightforward to implement, and has been shown to provide similar DNN training performance on the MNIST dataset [8].

A second design choice is the range of distinct neuron activation values that need to be supported by the hardware. In a digital implementation this translates into the number of bits, which would have area implications depending on the amount of local storage required, as well as the resolution of any analog to digital conversion circuits used to convert signals from the crossbar array into those bits. In an analog implementation, this would directly translate into the resolution between analog voltage levels and/or time-steps. We have shown that as few as six distinct neuron levels is sufficient for the MNIST dataset [8], delivering 92% test accuracy on a small three-layer network trained with only 5,000 examples, for which test accuracy with a perfect system can only reach 94%.

For a chip that only performs forward inference, no further steps are required. The excitations of the final output neurons can be output as the network's "guess" as to the classification of the data example that was input. In addition, the local storage of the intermediate neuron excitations, x, can be skipped. This then permits a significant degree of pipelining for forward-inference-only, with each array computing its forward inference for a different data-example, allowing the system to generate one new classification result for each time period equal to the time needed for the forward propagation of one layer.

Backpropagation of errors

Reverse propagation, or backpropagation of errors, is similar to forward propagation, but proceeds from output/hidden neurons back to the preceding hidden neurons (reversing the path followed by forward propagation). The quantity δ, known as the correction or error, together with the forward-propagated neuron activations x, are needed in order to perform the weight updates for neural network

training. At the output layer, these correction values are obtained by simple subtraction between the forward inference "guess" of the DNN and the correct labels. (In some cases, the DNN "guess" is first normalized by a softmax() operation, so that the raw value of the *j*th output neuron, z_j, is replaced by $\exp(z_j) / \sum \exp(z_k)$, where the denominator is summed over all *K* output neurons.)

Reverse propagation proceeds from the output neurons back toward the input neurons. At each array-block, a multiply-accumulate operation is again performed, but in a direction orthogonal to the integration performed during the forward-propagate step (e.g., along rows). Thus, the peripheral circuitry needs from Figure 3 are all present again, except that the locations are swapped. Thus, signals must be received, stored locally (as a few bits of δ information), and fed to column drivers on the "South side" of the array. Similarly, integration of the aggregate read current along each row occurs onto a capacitor, which is now located at the "West side" of each row, followed by measurement and transmission from shared "West side" circuitry. The need to "read" the array in two orthogonal directions at different times is a key distinction from conventional memory. With careful design of bias scheme and peripheral circuitry, it is possible to accomplish this without adding any more access transistors into the array. This parallel read operation is also subject to the same aggregate current and power constraints as the forward propagate parallel read.

For reverse propagation, because δ values are signed and because G^+ and G^- conductances share a common row, we must perform the integration in two time-multiplexed phases. First, positive signals are integrated onto the "West side" capacitor, representing the combination of positive δ values read through G^+ conductances and negative δ values read through G^- conductances. Then, the circuitry on the "West side" is reconfigured so that aggregate read current subtracts from the capacitor voltage, allowing contributions from positive δ values read through G^- conductances and negative δ values read through G^+ conductances. (Note that one could choose to provide differentiated row-lines for G^+ and G^- and a common column-line, but while that would allow the integration within this reverse-propagation operation to be performed in one phase, now the integration within forward-propagation would need to become a two-phase operation.)

Another important distinction from forward propagation is that a nonlinear squashing function is not applied. Instead, the multiply-accumulated sum (corresponding to the analog voltage on each West-side capacitor after integration) needs to be scaled by the derivative of the activation function, as evaluated at the neuron activation value. An exact tanh() or logistic derivative is not efficient to compute and multiply. Fortunately, a step-function derivative with two distinct states can be used. The necessary single bit, encoding whether the *x* value for this neuron during forward-propagate called for a large or small derivative, can readily be preserved as one of the few bits stored locally during forward propagation.

Multiplication by derivative values of zero and one is fairly straightforward to implement in hardware. This corresponds to simply enabling or disabling the transmission of an accumulated sum-of-deltas from any neuron stage to the preceding stage. With analog circuitry, more flexible multiplication by arbitrary scale factors can also be performed.

Once delta values have been integrated for a neuron layer, that layer is ready for weight update. This step can be pipelined with backward propagation of δ through additional upstream layers. Note that backward propagation from the first hidden layer back to the input neurons can be performed, but is not required. (The input neurons have no upstream synapses, so the highest layer that ever uses $\boldsymbol{\delta}$ values is the first hidden layer.)

Synaptic weight update

For weight-update, the few-bits that were stored locally at the "West side" of each row during forward-propagate, and the few-bits that were stored locally at the "South side" of each column are finally used. As shown in **Figure 4**, these bits are used to drive the row and column for write operations. (In the context of PCM, these would be partial-SET pulses, intended to increase conductance by a small portion of the total dynamic range between RESET and SET.) As described in [2], only the overlap of both the row and column pulse should lead to programming. The scheme described in [2] called for four timeslots (A, B, C, and D) so that the bit-pattern stored locally need only encode whether that row (or column) should participate in each of these timeslots. This mapping from the magnitude "Received signal" in Figure 3 to the particular bit-pattern stored for weight-update occurs during forward-propagate. This was also the point in time at which an effective "learning rate," η, can be imposed, in determining the threshold values at which the received *x* (or δ) transitions from requiring no programming pulses (few-bit pattern is all zeros) to one programming pulse (a timeslot is 1, others are 0), and so on.

In order to minimize excess CV^2 energy caused by unnecessary up-and-down toggling of the long row-lines and column-lines, the 1-bits in each few-bit pattern should be contiguous. Note that even a single timeslot provides quite similar DNN training performance, at least on the MNIST dataset [8]. In the case of a unidirectional NVM, the sign of delta dictates whether programming pulses are sent to the G^+ or G^- conductances. In the case of a bidirectional NVM, two programming phases (a first set of A, B, C, D timeslots, followed by a second set) can be used for improved performance [4], so that every weight update is implemented by programming both the G^+ and G^- conductances. Here, it is important to use an NVM that

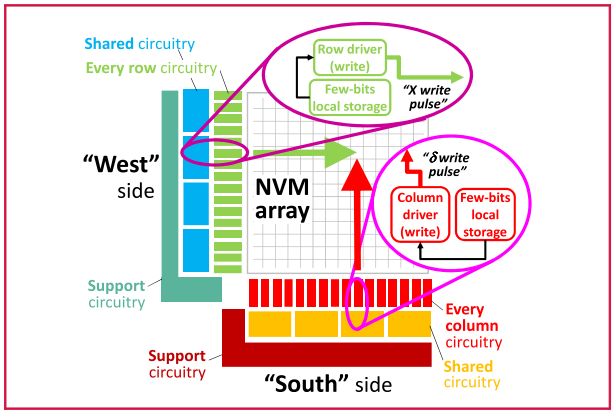

Figure 4

For weight-update, the few-bits stored locally at the "West side" of each row and the "South side" of each column are used to drive the row and column for write operations. (In the context of PCM, these would be partial-SET pulses, intended to increase conductance by a small portion of the total dynamic range between RESET and SET.) As described in reference [2], only the overlap of both the row and column pulse should lead to programming. The scheme described in reference [2] called for four timeslots (A, B, C, and D) so that the bit-pattern stored locally need only encode whether that row (column) should participate in each of these timeslots. In order to minimize excess CV^2 energy caused by unnecessary up-and-down toggling of the long row- and column-lines, the 1-bits in each few-bit pattern should be contiguous. Note that even a single timeslot provides quite similar DNN training performance, at least on the MNIST dataset [8]. In the case of a unidirectional NVM, the sign of delta dictates whether programming pulses are sent to the G^+ or G^- conductances. In the case of a bidirectional NVM, two phases can be used for improved performance [4], so that weight changes are implemented by programming both the G^+ and G^- conductances.

supports short writing times, such as the 25 ns partial-SET pulses [4] that can be used with PCM. Much longer write times would either slow the system, or require larger arrays or larger peripheral circuitry capable of aggregating multiple weight updates together.

Note that, even for low values of the learning rate, a given data example may require a large number of programming pulses to be applied to the array, perhaps exceeding the capabilities of the write drivers. This is particularly likely early in training. One approach that can work here is to write one stripe of the array-block at a time, enabling just the first 32 (or 64) columns, then the next, and so on. Such a procedure could be used early during training, and then turned off to allow subsequent training to occur even more rapidly. As a result, it will be important that the few-bits stored locally can be preserved for subsequent re-use, so that a single local-enable bit can control whether write drivers are enabled. Note that any sequencing of stored bits across timeslots should only be performed by support circuitry if that row or column line is enabled for write, in order to avoid wasted power. The simplest approach is to avoid the need for sequencing by using just a single timeslot bit. This has been shown to work just as well as four timeslots on both the MNIST and the MNIST + background noise datasets [8].

Occasional RESET

For NVM devices exhibiting a pronounced asymmetry in their conductance response, as mentioned earlier in this paper, an "Occasional RESET" step is necessary to allow accurate DNN training. Without this step, synapses receiving a large number of programming pulses inevitably move to the far right side of the "G-diamond," [2, 3] where both conductances approach their maximum value, and weight values are steered toward zero. This invariably causes networks to "freeze out," as the network stops training (because the δ values received by all upstream synaptic layers are scaled to near-zero by the low weight values) while it also steadily deletes the weights it has learned.

Although we refer to this process as "Occasional RESET" because the SET step in PCM is gradual, and the RESET step is abrupt, the exact same requirement would be necessary for a filamentary RRAM such as HfOx or TaOx RRAM. The only difference would be that in such devices, it is the RESET step that is gradual, and the SET step which is abrupt.

During this "Occasional RESET" step, DNN training is paused, and all conductances must be measured. This must necessarily be done in a row by row fashion. The measurements can be performed with the same integration circuitry described in Figure 3, except that the signals will be smaller (requiring longer integrations) and the integration of G^+ and G^- performed in sequence so that each conductance value can be independently obtained. One approach here might be to repurpose neighboring capacitors via current mirrors so that the weight, G^+ and G^- could all be obtained simultaneously.

Once the individual conductances are measured, any synapse for which both conductances are large can be flagged for RESET and incremental SET. Again, the few bits of local storage at each column can be used to hold these decisions, while the few bits at each row encode which row is being processed. This should allow the same write circuitry to perform both the RESET and subsequent SET operations. If verify should be necessary, the presence of the original weight as an analog voltage on one of the "South side" column-capacitors can be used to decide when to terminate the verify-SET loop.

Reconfigurability and directed redefinition of weights

As described earlier, it is important that such a single piece of hardware for on-chip learning support as wide a variety of DNN applications as possible. Thus a flexible routing network is essential to supporting networks that are very wide and networks that are very deep in a reconfigurable way. As such, an external user would experience no

constraints associated with the actual size of any one array block. Future work will address the possibility of adapting the architecture described here to support convolutional neural networks.

One important architecture in the training of extremely large DNN models involves the use of "data parallelism." Here a large number of processing nodes train the same DNN model in parallel, each using a different portion of the overall training database. As such, the weights of each copy of the DNN model tend to diverge based on their unique sequence of data examples. In order to leverage the large dataset, a "parameter server" is responsible for coordinating the work of the various processing nodes, by maintaining a single set of coordinated weights for the DNN.

The parameter server performs this work by alternating between sending out the current copy of these coordinated weights, and updating this set of weights with periodic updates sent back to the parameter server from each processing node. Such an approach can provide large speed-ups on the training of DNN models—for instance, speed-ups of $40\times$ have been measured for large models split across 50 processing nodes [29].

In order for any learning accelerator chip using NVM-based synapses to also support this kind of speedup through data parallelism, it is essential to support two additional chip modes, as shown at the left edge of Figure 2—weight overrides IN, and updated weights OUT. These two steps are made much more straightforward if the chip maintains a copy of the last "best" set of weights received from the parameter server. This can either be stored in digitized format at the edge of the chip, or more compactly within array-blocks that are NOT subsequently used for DNN training.

The process of receiving weight overrides then simply requires row-by-row tuning of the conductance values within both analog copies of the weight. The process of determining which weights have been modified by this chip requires only a simple row-by-row comparison between the two analog copies of the weights. Alternatively, if digital storage is available at the edge of the chip, or on a DRAM (dynamic random access memory) chip located nearby, then all the weights can be read out and digitized at the edge of the chip, allowing off-chip comparison between the last "best set" of weights and the chip's current set of weights, thus minimizing the network bandwidth needed for conveying weight updates from this processing node back to the parameter server. In this scenario, the near-off-chip digital copy of the "best set" of weights can be used to identify just those weights in need of override within the NVM-based chip.

Conclusion

We have described a generic architecture for accelerating the backpropagation-based training of DNNs by means of on-chip learning built around analog multiply-accumulate operations implemented with large NVM arrays. Peripheral circuitry on such an on-chip accelerator will need to support standard functionality for forward propagation, reverse propagation, and weight update. We presented circuit-level constraints and design choices toward realization of these functionalities. We also outlined some additional modes of operation (e.g., Occasional RESETs for certain NVMs, support for reconfigurable and flexible routing, and weight overrides). In all cases, we discussed tradeoffs and design choices that impact the acceleration factor and power consumption. These include several approximations to reduce the area overhead of peripheral circuitry, including approximate implementations of the squashing function and derivative, approximate crossbar-compatible weight update, sharing of circuitry between occasional RESET and parallel read/write modes, and reduction in the number of updated weights. Such practical circuit approaches will be of critical importance in the design of a practical dedicated on-chip training accelerator based on NVMs. Eventually, the mixed analog-digital-NVM design approach described here, once successful in implementing supervised on-chip learning for established algorithms such as backpropagation, might eventually lead to hardware implementations of much less-mature but sparse (and thus inherently more energy-efficient) spike-based algorithms offering one possible path toward continuous, online learning.

Acknowledgments

We acknowledge management support from Bülent Kurdi, Winfried Wilcke, Chung Lam, Spike Narayan, T. C. Chen, Wilfried Haensch, Heike Riel, and Dario Gil.

References

1. P. A. Merolla, J. V. Arthur, R. Alvarez-Icaza, A. S. Cassidy, J. Sawada, F. Akopyan, B. L. Jackson, N. Imam, C. Guo, Y. Nakamura, B. Brezzo, I. Vo, S. K. Esser, R. Appuswamy, B. Taba, A. Amir, M. D. Flickner, W. P. Risk, R. Manohar, and D. S. Modha, "A million spiking-neuron integrated circuit with a scalable communication network and interface," *Science*, vol. 345, no. 6197, pp. 668–673, 2014.
2. G. W. Burr, R. M. Shelby, C. di Nolfo, J. W. Jang, R. S. Shenoy, P. Narayanan, K. Virwani, E. U. Giacometti, B. Kurdi, and H. Hwang, "Experimental demonstration and tolerancing of a large-scale neural network (165,000 synapses), using phase-change memory as the synaptic weight element," in *Proc. Int. Electron Devices Meeting*, 2014, pp. 29.5.1–29.5.4.
3. G. W. Burr, R. M. Shelby, S. Sidler, C. di Nolfo, J. Jang, I. Boybat, R. S. Shenoy, P. Narayanan, K. Virwani, E. U. Giacometti, B. Kurdi, and H. Hwang, "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. Device*, vol. 62, no. 11, pp. 3498–3507, Nov. 2015.
4. G. W. Burr, P. Narayanan, R. M. Shelby, S. Sidler, I. Boybat, C. di Nolfo, and Y. Leblebici, "Large–scale neural networks implemented with nonvolatile memory as the synaptic weight element: comparative performance analysis (accuracy, speed, and power)," in *Proc. Int. Electron Devices Meeting*, 2015, pp. 4.4.1–4.4.4.
5. J.-W. Jang, S. Park, G. W. Burr, H. Hwang, and Y.-H. Jeong, "Optimization of conductance change in Pr1-xCaxMnO3–based

synaptic devices for neuromorphic systems," *IEEE Electron Device Lett.*, vol. 36, no. 5, pp. 457–459, May 2015.
6. S. Sidler, I. Boybat, R. M. Shelby, P. Narayanan, J. Jang, A. Fumarola, K. Moon, Y. Leblebici, H. Hwang, and G. W. Burr, "Large-scale neural networks implemented with non-volatile memory as the synaptic weight element: Impact of conductance response," in *Proc. 46th Eur. Solid-State Device Res. Conf.*, 2016, pp. 440–443.
7. A. Fumarola, S. Sidler, P. Narayanan, J. Jang, K. Moon, R. M. Shelby, H. Hwang, and G. W. Burr, "Accelerating machine learning with non-volatile memory: Exploring device and circuit tradeoffs," in *Proc. Int. Conf. Rebooting Comput.*, pp. 1–8, 2016.
8. P. Narayanan, L. L. Sanches, A. Fumarola, R. M. Shelby, J. Jang, H. Hwang, Y. Leblebici, and G. W. Burr, "Reducing circuit design complexity for neuromorphic machine learning systems based on non-volatile memory arrays," *Proc. IEEE Int. Symp. Circ. Sys.*, May 2017.
9. G. Cauwenberghs, C. F. Neugebauer, and A. Yariv, "Analysis and verification of an analog VLSI incremental outer-product learning system," *IEEE Trans. Neural Netw.*, vol. 3, no. 3, pp. 488–497, May 1992.
10. C. Schneider and H. Card, "Analogue CMOS Hebbian synapses," *Electron. Lett.*, vol. 27, no. 9, pp. 785–786, 1991.
11. R. Hasan and T. M. Taha, "Enabling back propagation training of memristor crossbar neuromorphic processors," in *Proc. Int. Joint Conf. Neural Netw.*, 2014, pp. 21–28.
12. M. Prezioso, F. Merrikh-Bayat, B. D. Hoskins, G. C. Adam, K. K. Likharev, and D. B. Strukov, "Training and operation of an integrated neuromorphic network based on metal-oxide memristors," *Nature*, vol. 521, no. 7550, pp. 61–64, 2015.
13. B. Reagen, P. Whatmough, R. Adolf, S. Rama, H. Lee, S. K. Lee, J. M. Hernández-Lobato, G.-Y. Wei, and D. Brooks, "Minerva: Enabling low-power, highly-accurate deep neural network accelerators," in *Proc. 43rd Int. Symp. Comput. Archit.*, 2016, pp. 267–278.
14. P. Chi, S. Li, Z. Qi, P. Gu, C. Xu, T. Zhang, J. Zhao, Y. Liu, Y. Wang, and Y. Xie, "PRIME: A novel processing-in-memory architecture for neural network computation in ReRAM-based main memory," in *Proc.43rd Annu. Int. Symp.Comput. Archit.*, 2016, vol. 43, pp. 27–39.
15. A. Shafiee, A. Nag, N. Muralimanohar, R. Balasubramonian, J. P. Strachan, M. Hu, R. S. Williams, and V. Srikumar, "ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars," in *Proc. 43rd Annu. Int. Symp. Comput. Archit.*, 2016, pp. 14–26.
16. S. Yu, P.-Y. Chen, Y. Cao, L. Xia, Y. Wang, and H. Wu, "Scaling-up resistive synaptic arrays for neuro-inspired architecture: Challenges and prospect," in *Proc. IEEE Int. Electron Devices Meet.*, 2015, pp. 17.3.1–17.3.4.
17. L. Gao, I.-T. Wang, P.-Y. Chen, S. Vrudhula, J.-S. Seo, Y. Cao, T.-H. Hou, and S. Yu, "Fully parallel write/read in resistive synaptic array for accelerating on-chip learning," *Nanotechnology*, vol. 26, no. 45, 2015, Art. no. 455204.
18. D. Negrov, I. Karandashev, V. Shakirov, Yu Matveyev, W. L. Dunin-Barkowski, and A. Zenkevich, "An approximate backpropagation learning rule for memristor based neural networks using synaptic plasticity," *J. Neurocomp.*, vol. 237(C), pp. 193–199, 2017.
19. S. B. Eryilmaz, S. Joshi, E. Neftci, W. Wan, G. Cauwenberghs, and H-S. P. Wong, "Neuromorphic architectures with electronic synapses," in *Proc. 17th Int. Symp. Quality Electron. Des.*, 2016, pp. 118–123.
20. T. Gokmen and Y. Vlasov, "Acceleration of deep neural network training with resistive cross-point devices: Design considerations," *Frontiers Neurosci.*, vol. 10, 2016, Art. no. 333.
21. D. Soudry, D. Di Castro, A. Gal, A. Kolodny, and S. Kvatinsky, "Memristor-based multilayer neural networks with online gradient descent training," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 26, no. 10, pp. 2408–2421, 2015.
22. G. Indiveri, B. Linares-Barranco, T. J. Hamilton, A. Van Schaik, R. Etienne-Cummings, T. Delbruck, S. C. Liu, P. Dudek, P. Häfliger, S. Renaud, and J. Schemmel, "Neuromorphic silicon neuron circuits," *Frontiers Neurosci.*, vol. 5, 2011, Art. no. 73.
23. S. Kim, M. Ishii, S. Lewis, T. Perri, M. BrightSky, W. Kim, R. Jordan, G.W. Burr, N. Sosa, A. Ray, J.-P. Han, C. Miller, K. Hosokawa, and C. Lam, "NVM neuromorphic core with 64k-cell (256-by-256) phase change memory synaptic array with on-chip neuron circuits for continuous in-situ learning," in *Proc. IEEE Int.Electron Devices Meeting*, 2015, pp. pp. 17.1.1–17.1.4.
24. R. A. Nawrocki, R. M. Voyles, and S. E. Shaheen, "A mini review of neuromorphic architectures and implementations," *IEEE Trans. Electron Devices*, vol. 63, no. 10, pp. 3819–3829, Oct. 2016.
25. G. W. Burr, R. S. Shenoy, K. Virwani, P. Narayanan, A. Padilla, B. Kurdi, and H. Hwang, "Access devices for 3D crosspoint memory," *J. Vacuum Sci. Technol. B*, vol. 32, no. 4, 2014, Art. no. 040802.
26. D. Rumelhart, G. E. Hinton, and J. L. McClelland, "A general framework for parallel distributed processing," in *Parallel Distributed Processing*, Cambridge, MA, USA: MIT Press, 1986.
27. Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.
28. P.-Y. Chen, B. Lin, I-T. Wang, T.-H. Hou, J. Ye, S. Vrudhula, J.-S. Seo, Y. Cao, and S. Yu, "Mitigating effects of non-ideal synaptic device characteristics for on-chip learning," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Des.*, 2015, pp. 194–199.
29. M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mane, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viegas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous distributed systems," arxiv:1603.04467, 2016.

Received September 2, 2016; accepted for publication October 1, 2016

Pritish Narayanan *IBM Research, San Jose, CA 95120 USA (pnaraya@us.ibm.com).* Dr. Narayanan received his Ph.D. degree in electrical and computer engineering from the University of Massachusetts–Amherst, in 2013. He joined IBM Research as a Research Staff Member. His current research interests include hardware systems for machine learning and cognitive computing.

Alessandro Fumarola *IBM Research, San Jose, CA 95120 USA (alessandro.fumarola@epfl.ch).* Mr. Fumarola is a student in the master's program of "Nanotechnology for the ICTs" held jointly by Politecnico Di Torino (Turin), PHELMA (Grenoble), and EPFL (Lausanne). He received his master's degree in 2016, after obtaining a bachelor's degree in industrial and power engineering from Politecnico di Torino. Mr. Fumarola was an intern at IBM Research - Almaden from March to August 2016, where he worked on cognitive computing with nonvolatile memory devices.

Lucas L. Sanches *IBM Research, San Jose, CA 95120 USA (lucas.lancellotti.sanches@usp.br).* Mr. Sanches is pursuing a B.S. degree in computer engineering from University of Sao Paulo (Brazil) for 2017. He was a Special Student at Massachusetts Institute of Technology during the academic year of 2015/2016. Mr. Sanches is a student member of the Institute of Electrical and Electronics Engineers and the Society for Industrial and Applied Mathematics. He was an intern in the Science and Technology department at IBM Research - Almaden during Summer 2016.

Kohji Hosokawa *IBM Research, Shin-Kawasaki, Kanagawa 2120032, Japan (khosoka@jp.ibm.com).* Mr. Hosokawa is a Senior Technical Staff Member in the Science and Technology department at IBM Research - Tokyo. He received a B.S. degree in electrical

engineering from Horoshima University. In 1983, he joined IBM Japan Yasu Application Laboratory where he developed various DRAM products. In 1996, he joined a 256-Mb SDRAM development alliance in IBM East Fishkill and IBM Burlington. After returning to Japan, he served as a manager of ASIC (application-specific integrated circuit) Design Center Japan in the IBM Microelectronics Division. In 2012, he transferred to IBM Research - Tokyo, where he currently manages the Neuromorphic Devices department.

Scott C. Lewis *IBM Research, Thomas J. Watson Research Center, Yorktown Heights, NY 10598 USA (sclewis@us.ibm.com).* In 1968, Mr. Lewis graduated from MIT and joined the IBM team designing a 128-bit bipolar SRAM chip. In 1973, as lead designer, he began work on a 36-kb DRAM, which in 1975, was qualified and became IBM's first production DRAM. From then until retirement in 2005, he worked for IBM, usually as technical leader, on subsequent DRAM generations up through 256-Mb. In 2008, he rejoined IBM to work on phase change memory.

Robert M. Shelby *IBM Research, San Jose, CA 95120 USA (rshelby@us.ibm.com).* Dr. Shelby is currently a Research Staff Member with IBM Research - Almaden. He received his Ph.D. degree in chemistry from the University of California at Berkeley. He joined IBM in 1978 and is a Fellow of the Optical Society of America.

Geoffrey W. Burr *IBM Research, San Jose, CA 95120 USA (gwburr@us.ibm.com).* Dr. Burr received his Ph.D. degree in electrical engineering from the California Institute of Technology in 1996. He joined IBM Research - Almaden in 1996, where he is currently a Principal Research Staff Member. Dr. Burr's current research interests include nonvolatile memory and cognitive computing. He is a senior member of the Institute of Electrical and Electronics Engineers.