Compressed sensing with approximate message passing using in-memory computing

Manuel Le Gallo, Abu Sebastian, *Senior Member, IEEE,* Giovanni Cherubini, *Fellow, IEEE,* Heiner Giefers, *Senior Member, IEEE,* and Evangelos Eleftheriou, *Fellow, IEEE*

***Abstract*—In-memory computing is a promising non-von Neumann approach where certain computational tasks are performed within resistive memory units by exploiting their physical attributes. In this paper, we propose a new method for fast and robust compressed sensing of sparse signals with approximate message passing recovery using in-memory computing. The measurement matrix for compressed sensing is encoded in the conductance states of resistive memory devices organized in a crossbar array. This way, the matrix-vector multiplications associated with both the compression and recovery tasks can be performed by the same crossbar array without intermediate data movements at potential $O(1)$ time complexity. For a signal of size N, the proposed method achieves a potential $O(N)$-fold recovery complexity reduction compared with a standard software approach. We show the array-level robustness of the scheme through large-scale experimental demonstrations using more than 256k phase-change memory devices.**

***Index Terms*—Approximate message passing, Compressed sensing, In-memory computing, Phase-change memory.**

I. Introduction

In-memory computing is an attractive approach for performing computationally expensive tasks of a high-level algorithm in an energy-efficient manner. For instance, crossbar arrays of resistive memory (memristive) devices can be used to store a matrix and perform analog matrix-vector multiplications at constant $O(1)$ time complexity without intermediate movements of data. This capability can be exploited in a wide range of applications from neural network inference to solving systems of linear equations [1]–[3].

Another well-suited application domain is that of complex optimization problems such as compressed sensing (CS) recovery. CS is an active research field in signal processing which attempts to perform sampling and compression simultaneously via a measurement matrix, and allows the recovery of a high-dimensional signal from low-dimensional noisy measurements. CS is used in various applications such as MRI, facial recognition, holography, audio restoration or in mobile phone camera sensors. In a camera sensor, the approach allows to significantly reduce the acquisition energy per image, or equivalently increase the image frame rate, by capturing only few measurements, e.g., 10%, instead of the whole image. However, CS recovery algorithms are usually complex, and conventional implementations are confronted with limited scalability owing to the large number of operations involved and high memory requirements. In-memory computing promises to significantly reduce the memory and computing resources needed to solve the problem as well as its computational complexity, at the cost of potentially reducing solution accuracy.

In Internet of Things (IoT) systems, it may be desirable to design implementations of CS with reconstruction on the same device, e.g., a sensor, using very low power, in order to have energy-efficient signal acquisition while at the same time not having to send the compressed signal to the cloud for reconstruction. Moreover, implementations of CS that can deal with very large measurement matrices may be desirable in applications where signals are received by large sensor arrays, as for example envisaged for the Square Kilometre Array [4], where the signal size may be on the order of 10^8.

In this paper, we propose an implementation of a CS recovery algorithm, namely Approximate Message Passing (AMP), based on memristive crossbar arrays, of which we presented a preliminary version in [5]. We experimentally investigate the impact of this memristive implementation on the performance of AMP, in particular on the reconstruction accuracy. The benefits and limitations of the memristive implementation are discussed for three use cases of the AMP algorithm, namely linear estimation, CS with soft-thresholding and compressive imaging with image denoising.

II. Overview of compressed sensing

A. Problem setting

The basic idea of CS is to acquire few sampling measurements from a high-dimensional signal, and subsequently to recover that signal accurately. The compressive measurements can be thought of as a linear mapping of a signal x_0 of length N to a measurement vector y of length $M < N$. If this process is linear, it can be modeled by a $M \times N$ measurement matrix A. The CS reconstruction problem is to determine the signal x_0 from the measurements y when sampled as

$$y = Ax_0 + w, \tag{1}$$

where w represents the measurement noise. CS asserts that signals can be recovered from fewer samples than dictated by the Shannon–Nyquist theorem if they are sparse, that is, if their information rate is lower than the Nyquist rate. If the signal x_0 is sparse in some transform domain, we can represent it as $x_0 = \Psi\xi$, where ξ contains only a few (k) non-negligible elements. It can be shown that if Ψ is incoherent with A, ξ can be recovered from y when $M < N$, as long as k is sufficiently

This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (grant agreement No. 682675). The results presented in this paper have been partially obtained in the framework of the project MNEMOSENE, which has received funding from European Union's Horizon 2020 research and innovation program (grant agreement No. 780215).

The authors are with IBM Research - Zurich, 8803 Rüschlikon, Switzerland (e-mail: anu@zurich.ibm.com, ase@zurich.ibm.com, cbi@zurich.ibm.com, hgiefers@gmail.com, ele@zurich.ibm.com).

small. Ψ represents the inverse transform matrix, for example an inverse Wavelet transform. CS is fundamentally different from transform coding, which is used for example in JPEG or MPEG compression. In the latter, the signal x_0 needs to be fully acquired, then the transform ξ is computed, and the largest k transform coefficients and their locations are kept so that the signal can be reconstructed. In CS, however, only $M < N$ measurements of x_0 are acquired while still being able to reconstruct the signal accurately. The downside is the cost of complex CS reconstruction algorithms.

In the case of a sparse signal x_0 and $w = 0$, a reconstruction of x_0 from y is obtained by solving the Basis Pursuit (BP) L_1 minimization problem. An alternative formulation known as Basis Pursuit Denoising (BPDN) extends BP to the more realistic noisy measurement case with $w \neq 0$. The solution of both BP and BPDN can be obtained by convex optimization using linear programming (LP) algorithms. However, the high computational complexity of LP represents an obstacle for the large problem sizes that occur very often in applications.

An appealing alternative to LP algorithms is offered by iterative thresholding algorithms, because of their low computational complexity. One particular iterative thresholding scheme to recover x_0 from y is of the form

$$\begin{aligned} x^{t+1} &= \eta_t(A^* z^t + x^t) \\ z^t &= y - Ax^t. \end{aligned} \tag{2}$$

Here, $x^t \in \mathbb{R}^N$ is the current estimate of x_0 at iteration t, $z^t \in \mathbb{R}^M$ is the current residual, and $\eta_t(\cdot)$ is a (typically nonlinear) function, A^* denotes the transpose of A and $x^0 = 0$. However, while offering low-complexity, the sparsity-undersampling tradeoff achieved by algorithm (2), that is, the smallest value that M can take given a certain sparsity of x_0 to successfully recover the signal, is usually less favorable than for LP-based reconstruction.

Recently, Donoho et al. proposed an AMP algorithm which adds a simple modification to (2) that substantially improves the sparsity-undersampling tradeoff without significantly increasing the computational complexity [6]. The AMP algorithm is formulated as [7]

$$\begin{aligned} x^{t+1} &= \eta_t(A^* z^t + x^t) \\ z^t &= y - Ax^t + \frac{N}{M} z^{t-1} \langle \eta'_{t-1}(A^* z^{t-1} + x^{t-1}) \rangle, \end{aligned} \tag{3}$$

where $\langle v \rangle \equiv N^{-1} \sum_{n=1}^{N} v_n$ denotes the average of a vector v, η'_t represents the derivative of η_t, $x^t \in \mathbb{R}^N$ is the current estimate of x_0 at iteration t, $z^t \in \mathbb{R}^M$ is the current residual, A^* denotes the transpose of A, and $x^0 = 0$. With respect to iterative thresholding (2), AMP includes the additional term $\frac{N}{M} z^{t-1} \langle \eta'_{t-1}(A^* z^{t-1} + x^{t-1}) \rangle$ in the computation of the residual, which is shown to substantially improve the sparsity-undersampling tradeoff [6]. AMP has the remarkable property that its solutions are governed by a state evolution whose fixed points (when unique) yield the true posterior means, in the limit $M, N \to \infty$, with the ratio M/N fixed, and assuming the elements of A are i.i.d. Gaussian random variables $A_{mn} \sim N(0, 1/M)$ [7].

B. *Compressed sensing hardware implementations*

Many works have focused on efficient hardware implementations for the acquisition of compressed measurements, such as in a camera sensor [8]–[10]. In an image sensor, the measurement matrix is typically binary and the measurement acquisition can be implemented either in the optical domain [8] or on-chip [9], [10]. Efficient implementations of single-shot imaging have been demonstrated with scalability up to 256×256 pixels consuming less than $100\,\text{mW}$ of power and showing no loss in SNR compared to normal (not compressed) capture [10]. In such implementations, the reconstruction algorithm is typically not implemented on-chip and therefore reconstruction has to be done offline.

For CS reconstruction, a number of implementations have been reported on FPGAs and ASICs designs. ASICs implementations of the Orthogonal Matching Pursuit (OMP) algorithm [11] and of the AMP algorithm [12] have been presented, as well as FPGA implementations of both [13]. Very recently, an implementation of the second-order cone program (SOCP) recovery algorithm for CS based on memristive crossbar arrays has been proposed [14], however without experimental validation.

In this work, we propose an implementation of the AMP algorithm based on memristive crossbar arrays, whereby the memristive arrays are used to perform the required matrix-vector multiplications. We aim to provide a robust set of experimental results of this implementation using phase-change memory (PCM) arrays. In comparison to typical high-precision implementations on GPUs or FPGAs, reconstruction with a memristive implementation will exhibit lower accuracy. The expectation is that the energy efficiency and scalability of a memristive implementation will allow to deal with much larger signals than in a typical high-precision implementation, and will yield faster and low-power solutions, at the cost of a reduced reconstruction accuracy, which may however be considered acceptable in many applications.

III. Realization using in-memory computing

A. *Implementation of compressed sensing with AMP recovery using resistive memory arrays*

The key idea of realizing CS using in-memory computing relies on the encoding of the elements of A as conductance values of memristive devices organized in a crossbar array, as depicted in Fig. 1a. One possible method to program the conductance values is by an iterative program-and-verify procedure. The compressed measurements (1) are acquired by applying x_0 as voltages to the crossbar rows via digital-to-analog conversion, and obtaining y through analog-to-digital conversion of the resulting output currents at columns. The positive and negative elements of A can be coded on separate devices together with a subtraction circuit, whereas negative vector elements can be applied as negative voltages.

Once the matrix A has been programmed in the crossbar array and the measurements y have been obtained, the AMP algorithm can be implemented as illustrated in Fig. 1b. The AMP algorithm is run in a dedicated processing unit, whereas the computation of $q^t = Ax^t$ and $u^t = A^* z^t$ is performed using

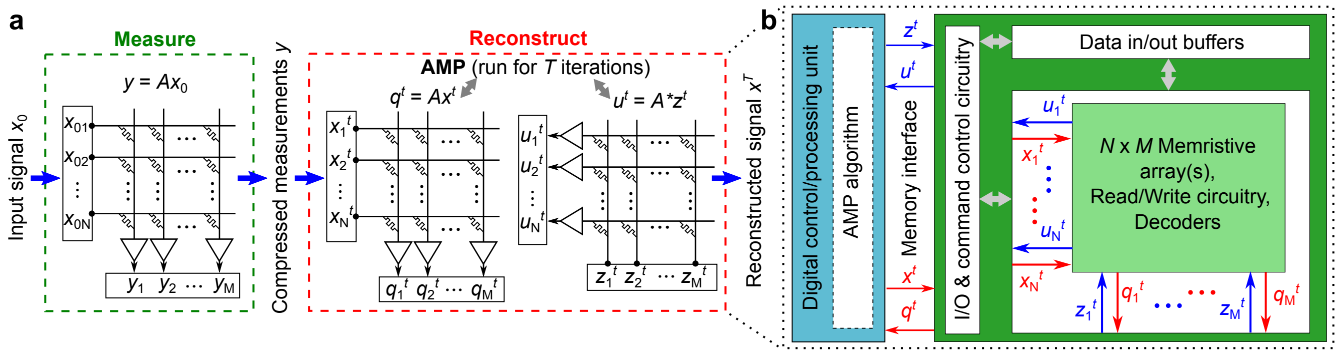
a
Measure
y = Ax0
Input signal x0
Compressed measurements y
Reconstruct
AMP (run for T iterations)
q^t = Ax^t
u^t = A*z^t
Reconstructed signal x^T
b
Digital control/processing unit
AMP algorithm
Memory interface
I/O & command control circuitry
Data in/out buffers
N x M Memristive array(s), Read/Write circuitry, Decoders

Fig. 1. (**a**) $N \times M$ memristive crossbar encoding the measurement matrix A used to acquire the CS measurements and to realize the matrix-vector computations of the AMP recovery algorithm, and (**b**) architecture of the memristive implementation of AMP.

the (same) crossbar array. The vector q^t is computed by applying x^t as voltages to the rows and reading back the resulting currents on the columns, and u^t by applying z^t as voltages to the columns and reading back the resulting currents on the rows. In a memristive crossbar, it has been argued that matrix-vector multiplications can be performed with constant time complexity $O(\gamma)$, where γ is independent of the crossbar size [3]. The reason is that the computation is performed in parallel through Kirchhoff's circuit laws locally at the same place where the matrix data is stored. Therefore, the complexity of (3) is potentially reduced from $O(MN)$ to $O(N)$ if A is dense, as it is the case for A with i.i.d. Gaussian elements. The precise value of γ will depend on the read current settling time and the time required to digitize the current by the peripheral circuitry. Consequently, larger crossbars may eventually lead to higher γ if some of the readout circuitry must be shared across columns/rows and multiplexed.

B. Physical implementation on prototype PCM chip

We implemented CS with AMP recovery using a prototype multi-level PCM chip that contains 1 million usable PCM cells. PCM is a resistive memory technology that is based on the rapid and reversible transition between the crystalline and amorphous phases of certain materials by application of suitable electrical pulses. Each PCM cell consists of a PCM device in series with an access transistor. The PCM devices are based on doped-$Ge_2Sb_2Te_2$ (d-GST) and are integrated into the prototype chip in 90 nm CMOS baseline technology [15]. In addition to the PCM cells, the prototype chip integrates the circuitry for cell addressing, on-chip analog-to-digital converter (ADC) for cell readout, and voltage- or current-mode cell programming. The PCM chip is interfaced to a hardware platform comprising two FPGA boards and an analog front-end (AFE) board. The layout, picture, and specifications of the experimental PCM chip with integrated read/write circuitry can be found in [5].

The selection of one PCM device is done by serially addressing a word line (WL) and a bit line (BL). For reading a PCM device, the selected BL is biased to a constant voltage (typically $0-300\,\mathrm{mV}$) by a voltage regulator via a voltage generated off-chip. The sensed current is integrated by a capacitor, and the resulting voltage is then digitized by the on-chip 8-bit cyclic ADC. The total time of one read is $1\,\mu$s. The readout characteristic is calibrated via on-chip reference polysilicon resistors. For programming a PCM device, a voltage generated off-chip is converted on-chip into a programming current. This current is then mirrored into the selected BL for the desired duration of the programming pulse. Each programming pulse is a box-type rectangular pulse (~ 1 ns rise/fall times) with a duration of 400 ns and an amplitude varying between 0 and $500\,\mu$A. Iterative programming involving a sequence of program-and-verify steps is used to program the PCM devices to the desired conductance values [16]. After each programming pulse, a verify step is performed and the value of the device conductance programmed in the previous iteration is read at a voltage of 0.2 V. The programming current applied to the PCM device in the subsequent iteration is adapted according to the sign of the value of the error between the target level and the read value of the device conductance. The total time of one program-and-verify step is approximately $2.5\,\mu$s. The array can be erased (RESET) using the maximum amplitude pulse of $500\,\mu$A and reprogrammed at will, and each cell can sustain approximately 10^9 programming pulses.

In our implementation of CS with AMP recovery, the element-by-element multiplications of the matrix-vector products were realized in the PCM chip, and the remaining operations were implemented in software. The elements of A were mapped to conductance values between 0 and $50\,\mu$S and programmed on 4 PCM devices averaged per element using iterative programming, with a conductance margin of $1.74\,\mu$S per device, that is, the iterative algorithm converges when the programmed conductance reaches a value within at most $1.74\,\mu$S from the target value. The matrix is programmed only once before CS is performed. Fig. 2a shows the number of programming cycles required and Fig. 2b-c show the conductance distributions for 5 representative levels. Here only 5 levels are shown for clarity, but in our experiments the conductance may assume any value in the range $0-50\,\mu$S. We mapped the vector elements to voltage values in the range $0-0.3$ V using a nonlinear mapping $f(V)$ to account for the slight nonlinearity of the current–voltage (I–V) characteristics of PCM devices [17]. The effect of this mapping is shown in Fig. 2d-e, where each point corresponds to the current of one PCM device measured at the applied voltage. The accuracy

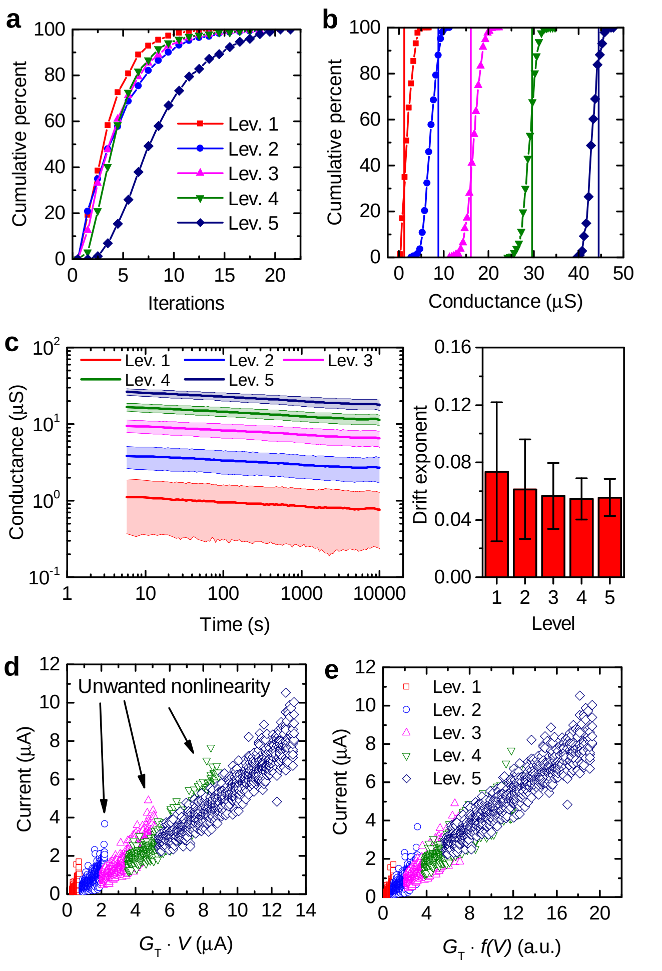

Fig. 2. Iterative programming of 5 representative conductance levels (vertical lines in **b**) on 5,000 devices of the PCM chip. (**a**) Number of iterations needed for convergence of the iterative programming algorithm; (**b**) conductance distributions at approx. $50\,\mu s$ after programming; (**c**) evolution of the mean conductance values of the 5 programmed levels vs. time; filled areas represent the standard deviation for each level; the plot on the right shows the calculated drift exponent ν of the 5 levels computed from $G(t) = G(t_0)(t/t_0)^{-\nu}$; (**d**) readout current of the 5,000 programmed PCM devices for a voltage range $0-0.3\,$V, plotted vs. $G_T \cdot V$, where V is the applied voltage and G_T is the target conductance of the different levels, and (**e**) readout current plotted vs. $G_T \cdot f(V)$, where $f(V) = V + 5V^3$.

of the matrix-vector computation with our PCM chip for a 256×256 matrix with i.i.d. Gaussian elements is comparable to that of a fixed-point implementation where the matrix and vector elements are quantized to 4 bits, as shown in Fig. 3.

To prevent errors in the multiplication results due to conductance drift of the PCM devices, we developed a drift calibration procedure which consists in periodically reading the summed current of L columns in the array during an experiment. Those L columns contain devices programmed to known conductance values $G_{mn}(t_0)$, therefore by reading them periodically at a constant voltage V_{cal} we can compensate for a global conductance shift as illustrated in Fig. 4. This procedure is especially simple because L can be chosen to be small, enough to get sufficient statistics, and the sum $\sum_{n=1}^{N}\sum_{m=1}^{L} G_{mn}(t_0)$ needs to be computed only once. The additional operations for drift calibration can be efficiently im-

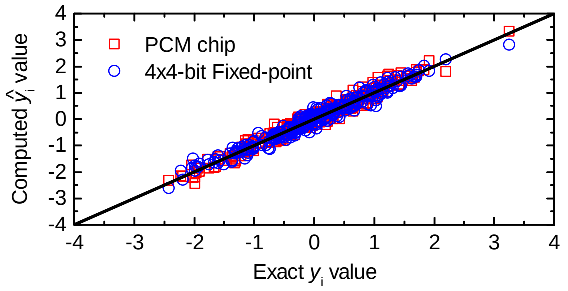

Fig. 3. Comparison of the precision in the computation of $y = Ax_0$ by the experimental PCM chip and 4×4-bit multiplications. A is a 256×256 Gaussian matrix coded in the PCM chip, x_0 is a 256-long Gaussian vector applied as voltages, and y_i is the i-th element of y.

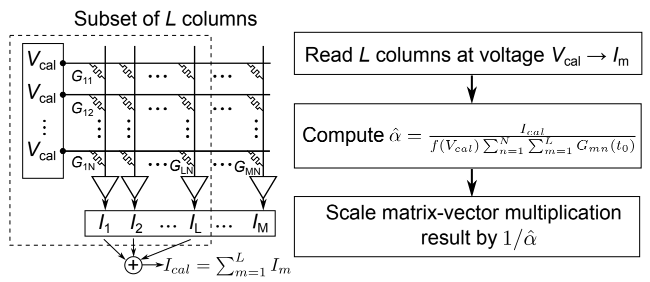
Subset of L columns
V_{cal}
G_{11}
G_{12}
G_{1N}
G_{LN}
G_{MN}
I_1 I_2 ... I_L ... I_M
$I_{cal} = \sum_{m=1}^{L} I_m$
Read L columns at voltage $V_{cal} \rightarrow I_m$
Compute $\hat{\alpha} = \frac{I_{cal}}{f(V_{cal})\sum_{n=1}^{N}\sum_{m=1}^{L} G_{mn}(t_0)}$
Scale matrix-vector multiplication result by $1/\hat{\alpha}$

Fig. 4. Calibration procedure to prevent errors due to conductance drift.

plemented and are not expected to incur significant time/power overhead. Reading the subset of L columns of the crossbar can be done while the PCM array is idle, i.e., when the digital unit performs the additional computations of the recovery algorithm, and additional means are needed to perform the L current summations as well as computing and storing $\hat{\alpha}$. They could be implemented either with on-chip digital circuitry or in the control/processing unit. In our experiments, the calibration procedure was performed in the control unit on $L = 40$ columns after every 5 matrix-vector multiplications.

IV. Experimental Results

A. Linear estimation

First, we study the simple use-case of linear estimation, where the vector x_0 is not sparse and its entries are i.i.d. Gaussian $N(0,1)$. In this case, the optimal AMP algorithm uses $\eta_t(x) = \lambda_t x$ with $\lambda_t = \frac{1}{1+\tau_t^2}$, where τ_t^2 is the variance of the empirical distribution of $A^* z^t + x^t - x_0$, which can be seen as the effective noise of the algorithm at iteration t [7]. τ_t^2 can be estimated by $\hat{\tau}_t^2 = \|z^t\|_2^2/M$, which is shown to be a good approximation of the variance of $A^* z^t + x^t - x_0$ in the large system limit [18].

We implemented this algorithm on the PCM chip for a random signal x_0 of size $N = 256$ and $M = N$ measurements. The $M \times N$ measurement matrix A was programmed in the PCM chip with i.i.d. Gaussian elements normalized such that the norm of its columns is approximately 1 [7]. The measurements y were obtained by applying x_0 as voltages on the PCM chip after matrix A had been programmed, thus realizing Ax_0 in hardware. Subsequently, x_0 was reconstructed with AMP using the PCM chip to compute the matrix-vector operations Ax^t

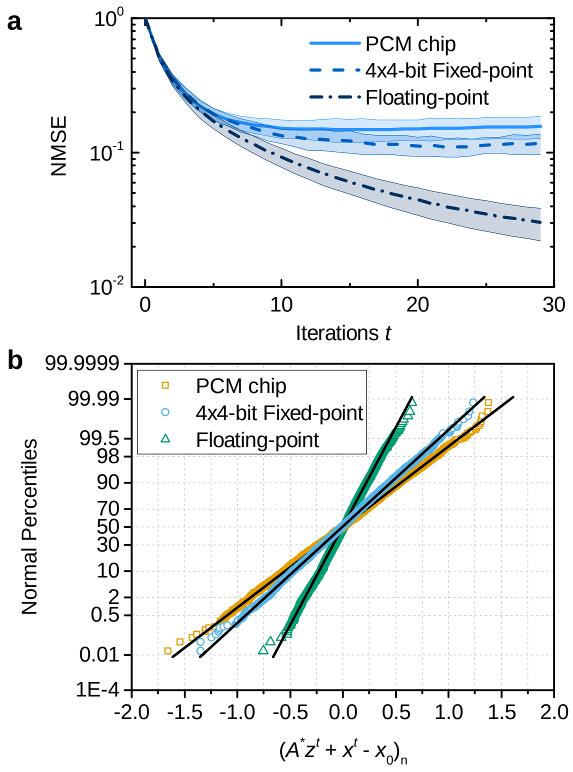

Fig. 5. (**a**) Normalized mean square error as a function of the number of AMP iterations for linear estimation with $N = M = 256$. The filled areas represent the standard deviation over 16 different realizations of A and x_0. (**b**) Empirical distribution of the effective noise $A^*z^t + x^t - x_0$ at the last AMP iteration $t = 29$ for the three implementations. All 16 experiments were used to build the empirical distributions.

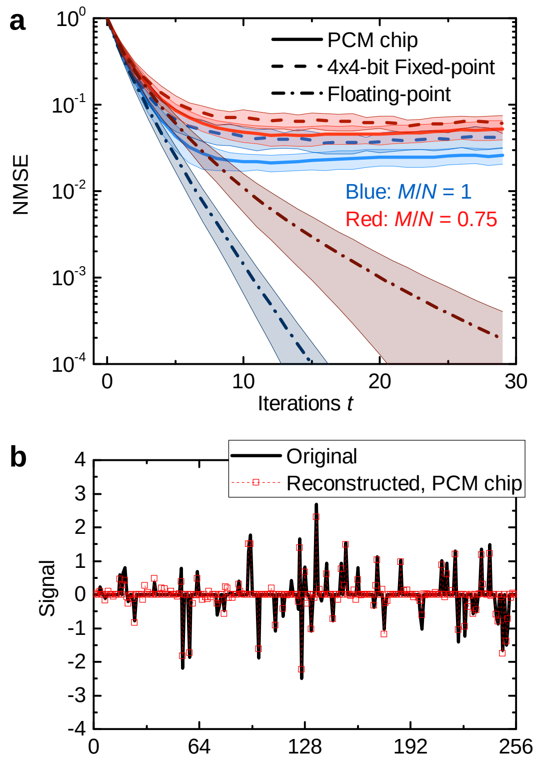

Fig. 6. (**a**) Normalized mean square error vs. number of AMP iterations for CS with soft-thresholding; filled areas represent the standard deviation over 16 different realizations of A and x_0, and (**b**) example of original and reconstructed signal for the PCM implementation with $M/N = 0.75$.

and A^*z^t, as shown in Fig. 1a. We performed the experiment 16 times for 16 different realizations of randomly generated A and x_0, and reported the mean and standard deviation of the normalized mean square error (NMSE) $\|x^t - x_0\|_2^2 / \|x_0\|_2^2$ over those 16 experiments. The different realizations of A and x_0 chosen were such that proper convergence of the AMP algorithm was obtained[1].

The evolution of the NMSE between the original and the reconstructed signal is shown in Fig. 5a. The NMSE decreases as $1/(1+t)$ for the floating-point implementation as dictated by state evolution [7]. For the PCM chip and an implementation where the multiplications in Ax^t and A^*z^t are done in 4×4-bit fixed-point arithmetic, the NMSE floors at values of approx. 0.15 and 0.12, respectively. However, the initial convergence rate of AMP is not affected by the inexact implementations. This finding will be further confirmed in the next experiments of Sections IV-B and IV-C.

An important feature of AMP is that the effective noise $A^*z^t + x^t - x_0$ is approximately Gaussian [18]. This allows the asymptotically exact analysis of AMP whereby the variance of this noise can be computed exactly from state evolution for any t when $N \to \infty$ [7]. Moreover, the variance can be used as input to the function η_t in order to optimally denoise this Gaussian noise [7]. For iterative thresholding (2), the effective noise is generally not Gaussian and state evolution does not hold [6], [7]. Hence, it is important to verify whether the Gaussianity of this noise is affected by the PCM implementation. We obtained the effective noise $A^*z^t + x^t - x_0$ at the last AMP iteration for the three implementations. We found no clear departure from a Gaussian distribution for both the PCM and fixed-point implementations, see Fig. 5b. The tails which deviate from an exact Gaussian distribution close to percentiles 0.01 and 99.99 observed in all three implementations are likely a consequence of the small system size ($N = 256$).

[1]Due to the small system size ($N = 256$), AMP does not converge properly for all combinations of randomly generated A and x_0. In the experiments, we ensured that for all realizations of A and x_0 chosen, the NMSE neither floors nor starts monotonically increasing in the floating-point implementation within the number of AMP iterations performed, in this case 29.

B. *Compressed sensing with soft-thresholding*

In this use case, the vector x_0 is k-sparse, i.e., it contains k non-zero elements, and its non-zero elements are i.i.d. Gaussian $N(0,1)$. In order to reconstruct x_0 from the measurements y, we use the AMP algorithm (3) with a sequence of soft-threshold functions $\eta_t(x)$ defined as [6]

$$\eta_t(x) = \begin{cases} x - \tau_t, & \text{if} \quad x > \tau_t \\ 0, & \text{if} \quad -\tau_t \le x \le \tau_t \\ x + \tau_t, & \text{if} \quad x < -\tau_t \end{cases} \tag{4}$$

with thresholds $\tau_t = \|z^t\|_2/\sqrt{M}$. For the soft-threshold function (4), the term $\frac{N}{M} z^{t-1} \langle \eta'_{t-1}(A^*z^{t-1} + x^{t-1}) \rangle$ in the AMP algorithm can be calculated explicitly and yields

$\frac{N}{M}z^{t-1}\langle\eta'_{t-1}(A^*z^{t-1}+x^{t-1})\rangle = \frac{1}{M}z^{t-1}\|\eta_{t-1}(A^*z^{t-1}+x^{t-1})\|_0$, where $\|x\|_0$ denotes the number of non-zero elements of x.

We performed experiments for a random signal x_0 of size $N = 256$ and $k = 64$ randomly distributed non-zero elements. We tested cases for sampling rates of $M/N = 1$ (no compression) and $M/N = 0.75$, each with 16 different realizations of randomly generated A and x_0. The evolution of the NMSE between the original and the reconstructed signal is shown in Fig. 6a. As in the previous use-case, the initial convergence rate of AMP is unaffected by the approximate multiplications done in the PCM chip and the magnitude of the NMSE floor obtained with the PCM chip is comparable to the 4×4-bit fixed-point implementation. When using a lower sampling rate $M/N = 0.75$, the convergence rate of AMP decreases and the NMSE floor increases for the inexact implementations compared to $M/N = 1$.

In certain applications, it is sufficient to recover only the sparsity pattern of x_0, without being concerned with the exact values of the non-zero elements. We show in Fig. 6b the original and reconstructed signals for one of the experiments performed with 0.75 sampling rate. We see that the general shape and the sparsity pattern of the signal are well recovered in the PCM implementation. Thus, in applications where the reconstruction accuracy is not of paramount importance, the accuracy obtained with our current prototype PCM chip may already be sufficient.

C. Compressive imaging with image denoising

Compressive imaging refers to performing CS on image signals. The elements of x_0 thus represent the pixel intensities of an image. The goal is to acquire the image with $M \ll N$ measurements, and to reconstruct it accurately. A general methodology for compressive imaging with AMP was recently introduced by Metzler et al. [18]. They developed an extension of the AMP algorithm that uses a denoiser within its iterations. The proposed algorithm is given by

$$\begin{aligned} x^{t+1} &= D_{\tau_t}(A^*z^t + x^t) \\ z^t &= y - Ax^t + \tfrac{1}{M}z^{t-1}\mathrm{div}D_{\tau_{t-1}}(A^*z^{t-1}+x^{t-1}) \\ \tau_t^2 &= \|z^t\|_2^2/M, \end{aligned} \tag{5}$$

where D_τ denotes a denoiser which takes as input a signal plus Gaussian noise and an estimate of the standard deviation of that noise τ, and $\mathrm{div}D_\tau(x) = \sum_{n=1}^{N}\frac{\partial D_\tau(x)_n}{\partial x_n}$ denotes the divergence of the denoiser, where $D_\tau(x)_n$ is the n-th element of $D_\tau(x)$ and x_n is the n-th element of x.

We tested this algorithm using the 128×128 pixel "house" image shown in Fig. 7b as signal x_0. We implemented two different denoisers:

- **Wavelet thresholding** transforms the signal into a wavelet basis, thresholds the coefficients, and then inverts the transform. If W denotes the wavelet transform, this denoiser is defined as $D_{\tau_t}(x) = W^{-1}\eta_t(Wx)$. We used the soft-threshold function (4) as η_t and 2D Haar wavelet transform. The divergence of this denoiser can be calculated explicitly and yields $\mathrm{div}D_{\tau_{t-1}}(A^*z^{t-1}+x^{t-1}) = \|\eta_{t-1}(W(A^*z^{t-1}+x^{t-1}))\|_0$, which is the number of non-zero elements of the thresholded sparsified estimate.

TABLE I
PSNR (IN DB) OF THE 128×128 "HOUSE" IMAGE RECONSTRUCTIONS.

	PCM chip	Fixed-point	Floating-point
Wavelet thresh.	27.15	27.39	32.50
BM3D	34.58	32.27	45.06

- **BM3D** (block matching 3D collaborative filtering) can be considered a combination of non-local means (averaging weighted neighboring pixels) and wavelet thresholding. The term $\mathrm{div}D_{\tau_{t-1}}(A^*z^{t-1}+x^{t-1})$ cannot be calculated explicitly and thus is estimated using the Monte-Carlo procedure described in Metzler et al. [18]. The divergence is estimated with $\mathrm{div}D_\tau(x) \simeq \frac{b^*}{\varepsilon}\left(D_\tau(x+\varepsilon b) - D_\tau(x)\right)$ for small ε and vector b with elements i.i.d. $N(0,1)$. BM3D performs much better on images than wavelet thresholding because images are not exactly sparse in the wavelet domain.

The length of x_0 in this experiment is $N = 16384$. For such a large value of N, it is not possible to code all elements of a $M\times N$ Gaussian matrix in our PCM hardware, which has only 1 million usable devices. To overcome this difficulty, we use a block-based compression approach, whereby a small measurement matrix H of size $M_s \times N_s$ is used, with $N_s = 256$. We perform measurements on consecutive 16×16 pixel blocks using the same measurement matrix, H. In order to obtain uncorrelated measurements and ensure the convergence of AMP, we perform a (fixed) random permutation P of the pixel intensities before doing the measurements. The matrix A can be thus written as $A = \mathrm{blkdiag}(H)P$, where $\mathrm{blkdiag}(H)$ is a $M\times N$ matrix with N/N_s main diagonal blocks matrices H, where it is assumed that N is a multiple of N_s and $M_s/N_s = M/N$. The elements of H are i.i.d. $\sim N(0, 1/M_s)$.

We programmed a 128×256 Gaussian measurement matrix H in the PCM chip (sampling rate $M/N = 1/2$), divided the image into 16×16 pixel blocks, and compressed each block individually with the PCM chip. Subsequently, the image was reconstructed with algorithm (5) using the PCM chip to compute the matrix-vector operations Ax^t and A^*z^t. In Fig. 7a, we show the NMSE evolution for the PCM, fixed-point, and floating-point implementations for wavelet thresholding and BM3D denoisers. The peak signal-to-noise ratio[2] (PSNR) at the last AMP iteration is reported in Table I. It can be seen that using a better denoiser (e.g. BM3D) results in a lower final NMSE in the PCM and fixed-point implementations. It indicates that denoisers can be used effectively to improve the reconstruction accuracy by mitigating the errors from the PCM chip. Moreover, the convergence rate of AMP is only affected by the choice of the denoiser, but not by the approximate implementations.

V. DISCUSSION

There are several reasons why AMP is well suited for a memristive implementation. First, matrix A does not change

[2] $\mathrm{PSNR} = 10\log_{10}\left(\frac{255^2}{\|\hat{x}-x_0\|_2^2/N}\right)$, where $\hat{x}$ is the estimate of x_0.

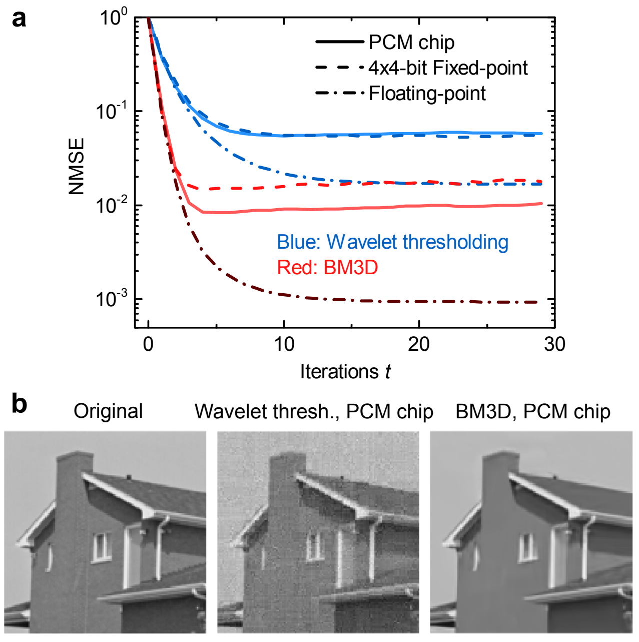

Fig. 7. (**a**) Evolution of the normalized mean square error in image reconstruction for wavelet thresholding and BM3D denoisers with $M/N = 1/2$, and (**b**) original and reconstructed images with the PCM implementation.

over iterations, thus only read operations are performed during AMP reconstruction. Therefore, matrix A needs to be programmed only once and will be retained in the array thanks to the non-volatility of the PCM devices. The read operations that are performed during reconstruction require significantly less power than programming, and thus can be heavily parallelized. With the 90-nm PCM technology used in this work, we estimate the read energy to be between 1 and 100 fJ per device depending on the programmed resistance state, compared with approximately 100 pJ for programming (assuming 5 program-and-verify iterations). Moreover, unlike programming endurance, the read endurance (at least in PCM) is essentially unlimited, hence this implementation is favorable with respect to device reliability issues and will not lead to device degradation due to excessive reprogramming at every iteration.

The effect of device imperfections and failures on the final reconstruction NMSE is discussed in [5]. We found that the AMP recovery can tolerate conductance variations due to programming errors (up to 20%), and up to 20% stuck-SET and stuck-RESET device failures. Device imperfections that have a detrimental effect on the reconstruction accuracy include device conductance noise (most dominant effect) and I–V nonlinearity. Finally, the achievable reconstruction NMSE is ultimately limited by the resolution of the DACs/ADCs used at the input/output of the crossbar array.

To quantify the potential energy gains of the memristive implementation over a digital design, based on the figures currently achieved with our prototype PCM chip, we made an FPGA design that operates at the same speed and same precision at which we expect a PCM-based crossbar to perform [5]. In (3), matrix-vector multiplications are the most expensive operations, so we compared the memristive crossbar analog multiplier with a 4-bit FPGA multiplier design. The 4-bit matrix elements are stored in the FPGA block-RAM, and 32 dot-product units operate in parallel to compute a 256×256 matrix-vector product in 1.2 μs. The dynamic power consumption achieved with this design is 800 mW [5]. In a 256×256 PCM-based crossbar, the dynamic power dissipation in the devices for one read operation would be on the order of 13.1 mW (read current of 1 μA per device at 0.2 V). Thus, a 256×256 PCM-based crossbar in 90-nm technology operating at 1 μs cycle time plus two 8-bit ADCs operating at 125 MS/s to convert the current (12 mW/GSps power consumption) is expected to consume 16.2 mW, which is 50 times less than the FPGA design. The power advantage arises because only read operations, which consume little energy, are performed in the memristive crossbar for multiplications.

While PCM devices were used for the experiments presented in this work, other memory devices could be considered to perform the analog matrix-vector multiplications in the proposed compressed sensing implementation. Potential candidates include metal-oxide RRAM [3], NOR Flash [19], and SRAM [20]. The main advantages of PCM for this application are its multi-level capability along with fast read/write latency and non-volatility, however the PCM programming current is generally higher than other technologies and resistance drift poses additional challenges that need to be addressed. Assessing different technologies for in-memory computing should account for array-level variability, device noise, and accuracy/ease of device programming in addition to latency and power consumption.

In the ASIC implementation of AMP reported in [12], the multiply-accumulate units (MAC) and the matrix generating unit take most of the chip area and are responsible for most of the power consumption, which amounts to $> 90\%$ in the proposed AMP-M design for arbitrary matrices. In such an implementation, matrix A would have to be explicitly stored (in off-chip DRAM) or its coefficient would have to be generated on the fly at every AMP iteration. In a memristive implementation, matrix A is stored in the memristive array(s) in a non-volatile manner, thus avoiding the need of a unit to generate its coefficients or using an off-chip DRAM, while still being able to reprogram it without redesigning the entire circuit. Moreover, by computing the matrix-vector multiplications inside the memristive array, the use of MAC units, which are expensive in both power and area when implemented in CMOS, is completely avoided.

Furthermore, a remarkable property of AMP is that its convergence rate is independent of the precision of the matrix-vector multiplications. This is a highly desirable property for this type of implementation, as the number of AMP iterations needed for reconstruction will not be larger than in a floating-point implementation. We also found that the NMSE floor due to computational errors can be lowered by using appropriate denoisers within AMP. Obviously, using a complex denoiser such as BM3D might not be efficient from an implementation point of view, because the speedup obtained by performing the matrix-vector multiplications in the memristive array may be overcompensated by the time required to apply the denoiser. However, an interesting avenue would be to design a denoiser that is specifically aimed at removing the computational errors from the memristive array.

Regarding the limitations of the memristive implementation, the computational errors from the memristive array are currently the biggest drawback. Very accurate reconstruction cannot be currently achieved with our prototype PCM chip, which performs with a precision similar to that of a matrix-vector product in 4×4-bit fixed-point implementation. However, the precision of analog in-memory computation is expected to improve as the technology matures, e.g., with concepts such as projected memory to reduce the noise and drift [21]. The precision could be further increased by mapping a single column of the matrix across multiple physical columns of an array encoding different bits and applying the input vector to the array one or several bits at a time, still performing in-memory computing, at the expense of area and energy penalty, and additional support required by the peripheral circuitry.

Another limitation is that, for CS applications, it might be hard to justify the memristive implementation versus a digital implementation with a 1-bit measurement matrix, as the latter shows no loss in SNR for the compressed measurement acquisition and no multipliers are needed for a binary matrix [10]. However, this type of implementation is limited to one specific application only, i.e., only a binary measurement matrix is supported, whereas a memristive implementation can be used for any arbitrary measurement matrix. Moreover, such efficient implementations currently only acquire the compressed measurements and do not support reconstruction, which has to be done off-chip. The attractiveness of the memristive implementation is that both compression and reconstruction could be done on the same platform.

VI. Conclusion

We propose an implementation of CS with AMP recovery based on memristive crossbar arrays. The measurement matrix elements are programmed as conductance values of memristive devices in crossbar arrays, which are used to perform the matrix-vector multiplications in both the compression and the recovery algorithm. In this way, the computational complexity of AMP recovery is potentially reduced from $O(MN)$ to $O(N)$. We tested this implementation experimentally for three use-cases of AMP using more than 256k PCM devices in a prototype multi-level PCM chip to perform the matrix-vector multiplications. We found that the convergence rate of AMP is not affected by performing matrix-vector multiplications in the PCM array. The accuracy achieved with our prototype PCM chip is comparable to that of a fixed-point implementation where the matrix and vector elements are quantized to 4 bits. In applications where the reconstruction accuracy is not of paramount importance, the memristive implementation could represent a viable solution to provide more efficient AMP reconstruction than a full von Neumann implementation.

Acknowledgment

We acknowledge N. Papandreou and U. Egger for experimental help, L. Kull and T. Toifl for discussions, and M. Brightsky for providing the PCM devices used in this work.

References

[1] M. Le Gallo, A. Sebastian, R. Mathis, M. Manica, H. Giefers, T. Tuma, C. Bekas, A. Curioni, and E. Eleftheriou, "Mixed-precision in-memory computing," *Nat. Electron.*, vol. 1, no. 4, pp. 246–253, April 2018, doi: 10.1038/s41928-018-0054-8.

[2] P. M. Sheridan, F. Cai, C. Du, W. Ma, Z. Zhang, and W. D. Lu, "Sparse coding with memristor networks," *Nat. Nanotechnol.*, vol. 12, no. 8, pp. 784–789, May 2017, doi: 10.1038/nnano.2017.83.

[3] M. Hu, C. E. Graves, C. Li, Y. Li, N. Ge, E. Montgomery, N. Davila, H. Jiang, R. S. Williams, J. J. Yang, Q. Xia, and J. P. Strachan, "Memristor-based analog computation and neural network classification with a dot product engine," *Adv. Mater.*, vol. 30, no. 9, p. 1705914, Jan 2018, doi: 10.1002/adma.201705914.

[4] G. Cherubini, P. Hurley, M. Simeoni, and S. Kazemi, "Imaging in radio interferometry by iterative subset scanning using a modified AMP algorithm," in *Proc. ICASSP*, March 2016, pp. 3326–3330, doi: 10.1109/ICASSP.2016.7472293.

[5] M. Le Gallo, A. Sebastian, G. Cherubini, H. Giefers, and E. Eleftheriou, "Compressed sensing recovery using computational memory," in *IEDM Tech. Dig.*, Dec 2017, pp. 28.3.1–28.3.4, doi: 10.1109/IEDM.2017.8268469.

[6] D. L. Donoho, A. Maleki, and A. Montanari, "Message-passing algorithms for compressed sensing," *Proc. Natl. Acad. Sci. USA*, vol. 106, no. 45, pp. 18 914–18 919, Nov 2009, doi: 10.1073/pnas.0909892106.

[7] M. Bayati and A. Montanari, "The dynamics of message passing on dense graphs, with applications to compressed sensing," *IEEE Trans. Inf. Theory*, vol. 57, no. 2, pp. 764–785, Feb 2011, doi: 10.1109/TIT.2010.2094817.

[8] M. F. Duarte, M. A. Davenport, D. Takbar, J. N. Laska, T. Sun, K. F. Kelly, and R. G. Baraniuk, "Single-pixel imaging via compressive sampling," *IEEE Signal Process. Mag.*, vol. 25, no. 2, pp. 83–91, March 2008, doi: 10.1109/MSP.2007.914730.

[9] L. Jacques, P. Vandergheynst, A. Bibet, V. Majidzadeh, A. Schmid, and Y. Leblebici, "CMOS compressed imaging by random convolution," in *Proc. ICASSP*, April 2009, pp. 1113–1116, doi: 10.1109/ICASSP.2009.4959783.

[10] Y. Oike and A. El Gamal, "CMOS image sensor with per-column $\sigma\delta$ ADC and programmable compressed sensing," *IEEE J. Solid-State Circuits*, vol. 48, no. 1, pp. 318–328, Jan 2013, doi: 10.1109/JSSC.2012.2214851.

[11] P. Maechler, P. Greisen, B. Sporrer, S. Steiner, N. Felber, and A. Burg, "Implementation of greedy algorithms for LTE sparse channel estimation," in *Proc. ASILOMAR*, Nov 2010, pp. 400–405, doi: 10.1109/ACSSC.2010.5757587.

[12] P. Maechler, C. Studer, D. E. Bellasi, A. Maleki, A. Burg, N. Felber, H. Kaeslin, and R. G. Baraniuk, "VLSI design of approximate message passing for signal restoration and compressive sensing," *IEEE Trans. Emerg. Sel. Topics Circuits Syst.*, vol. 2, no. 3, pp. 579–590, Sept 2012, doi: 10.1109/JETCAS.2012.2214636.

[13] L. Bai, P. Maechler, M. Muehlberghuber, and H. Kaeslin, "High-speed compressed sensing reconstruction on FPGA using OMP and AMP," in *Proc. ICECS*, Dec 2012, pp. 53–56, doi: 10.1109/ICECS.2012.6463559.

[14] S. Liu, A. Ren, Y. Wang, and P. K. Varshney, "Ultra-fast robust compressive sensing based on memristor crossbars," in *Proc. ICASSP*, March 2017, pp. 1133–1137, doi: 10.1109/ICASSP.2017.7952333.

[15] M. Breitwisch, T. Nirschl, C. Chen, Y. Zhu, M. Lee, M. Lamorey, G. Burr, E. Joseph, A. Schrott, J. Philipp *et al.*, "Novel lithography-independent pore phase change memory," in *VLSI Symp. Tech. Dig.*, June 2007, pp. 100–101, doi: 10.1109/VLSIT.2007.4339743.

[16] N. Papandreou, H. Pozidis, A. Pantazi, A. Sebastian, M. Breitwisch, C. Lam, and E. Eleftheriou, "Programming algorithms for multilevel phase-change memory," in *Proc. ISCAS*, May 2011, pp. 329–332, doi: 10.1109/ISCAS.2011.5937569.

[17] M. Le Gallo, M. Kaes, A. Sebastian, and D. Krebs, "Subthreshold electrical transport in amorphous phase-change materials," *New J. Phys.*, vol. 17, no. 9, p. 093035, Sept 2015, doi: 10.1088/1367-2630/17/9/093035.

[18] C. A. Metzler, A. Maleki, and R. G. Baraniuk, "From denoising to compressed sensing," *IEEE Trans. Inf. Theory*, vol. 62, no. 9, pp. 5117–5144, Sept 2016, doi: 10.1109/TIT.2016.2556683.

[19] X. Guo, F. M. Bayat, M. Bavandpour, M. Klachko, M. R. Mahmoodi, M. Prezioso, K. K. Likharev, and D. B. Strukov, "Fast, energy-efficient, robust, and reproducible mixed-signal neuromorphic classifier based on embedded NOR flash memory technology," in *IEDM Tech. Dig.*, Dec 2017, pp. 6.5.1–6.5.4, doi: 10.1109/IEDM.2017.8268341.

[20] A. Biswas and A. P. Chandrakasan, “Conv-RAM: An energy-efficient SRAM with embedded convolution computation for low-power CNN-based machine learning applications,” in *Proc. ISSCC*, Feb 2018, pp. 488–490, doi: 10.1109/ISSCC.2018.8310397.

[21] W. W. Koelmans, A. Sebastian, V. P. Jonnalagadda, D. Krebs, L. Dellmann, and E. Eleftheriou, “Projected phase-change memory devices,” *Nat. Commun.*, vol. 6, no. 8181, Sept 2015, doi: 10.1038/ncomms9181.