

# Differentiable Content Addressable Memory with Memristors


Giacomo Pedretti,\* Catherine E. Graves,\* Thomas Van Vaerenbergh, Sergey Serebryakov, Martin Foltin, Xia Sheng, Ruibin Mao, Can Li, and John Paul Strachan\*

Memristors, Flash, and related nonvolatile analog device technologies offer in-memory computing structures operating in the analog domain, such as accelerating linear matrix operations in array structures. These take advantage of analog tunability and large dynamic range. At the other side, content addressable memories (CAM) are fast digital lookup tables which effectively perform nonlinear Boolean logic and return a digital match/mismatch value. Recently, nonvolatile analog CAMs have been presented merging analog storage and analog search operations with digital match/mismatch output. However, CAM blocks cannot easily be inserted within a larger adaptive system due to the challenges of training and learning with binary outputs. Here, a missing link between analog crossbar arrays and CAMs, namely a differentiable content addressable memory (dCAM), is presented. Utilizing nonvolatile memories that act as a “soft” memory with analog outputs, dCAM enables learning and fine-tuning of the memory operation and performance. Four applications are quantitatively evaluated to highlight the capabilities: improved data pattern storage, improved robustness to noise and variability, reduced energy and latency performance, and an application to solving Boolean satisfiability optimization problems. The use of dCAM is envisioned as a core building block of fully differentiable computing systems employing multiple types of analog compute operations and memories.

G. Pedretti, C. E. Graves, T. Van Vaerenbergh, S. Serebryakov, M. Foltin, X. Sheng  
 Hewlett Packard Labs  
 Milpitas, CA 95035, USA  
 E-mail: giacomo.pedretti@hpe.com; catherine.graves@hpe.com

R. Mao, C. Li  
 The University of Hong Kong  
 Hong Kong SAR, China

J. P. Strachan  
 Peter Grünberg Institute (PGI-14), Forschungszentrum Jülich GmbH  
 Jülich, Germany 52428  
 RWTH Aachen University  
 52062 Aachen, Germany  
 E-mail: j.strachan@fz-juelich.de

 The ORCID identification number(s) for the author(s) of this article can be found under <https://doi.org/10.1002/aelm.202101198>.

© 2022 The Authors. Advanced Electronic Materials published by Wiley-VCH GmbH. This is an open access article under the terms of the Creative Commons Attribution-NonCommercial License, which permits use, distribution and reproduction in any medium, provided the original work is properly cited and is not used for commercial purposes.

DOI: 10.1002/aelm.202101198

## 1. Introduction

For specific tasks, in-memory and analog computing<sup>[1,2]</sup> has promise to save energy and time compared to conventional computing based on general-purpose von-Neumann architectures, due to significantly reduced data movement.<sup>[3]</sup> Accelerators based on nonvolatile memristive devices can offer high data density and analog operations, which can enable acceleration of many important computing tasks including neural network training and inference,<sup>[4–9]</sup> image processing,<sup>[10,11]</sup> optimization,<sup>[12–14]</sup> and algebraic problems.<sup>[15,16]</sup> These are based predominantly on cross-bars of memristive devices,<sup>[17]</sup> which can directly map from a target matrix due to the analog tunability of the programmable, nonvolatile resistive switches.<sup>[18]</sup> Going beyond this, another in-memory computing primitive is based on content addressable memory (CAM),<sup>[19]</sup> which operates in an opposite way as random access memory. Here, data is given as input and the location of a match to that data is returned, offering cross-memory lookup and pattern matching acceleration<sup>[20–23]</sup> in just a few clock cycles. This connects to the biologically-inspired concept of associative memories,<sup>[24–26]</sup> and the powerful notion of associative computing.<sup>[27–32]</sup> Today, the physical implementations of CAMs and Ternary CAM (TCAM)—which introduce a third wild card “x” state besides the usual “0” and “1”—are typically based on SRAM circuits, but have high area and cost.<sup>[33]</sup> Utilizing nonvolatile technology has been explored to reduce power and area, including with memristive devices<sup>[34–39]</sup> and ferroelectric technology.<sup>[40,41]</sup> Taking advantage of the analog tunability and dynamic range in such devices an analog (or multi-bit) CAM based on memristors<sup>[42]</sup> was introduced. The concept is to store ranges in each CAM cell using two memristors that define the boundaries, either in discrete allowed intervals (multi-bit) or continuously tunable (analog). Similarly, the inputs can take discrete values (multi-bit input) or be continuous. This can offer higher data density and lower energy per search,<sup>[42]</sup> but does become susceptible to variability and fluctuations in memristive devices which continue to mature. Additionally, unique operations such as inequality tests can be conducted with an analog CAM. This has been described and utilized to accelerate the decisions in tree-based machine learning models, including random forests.<sup>[43]</sup> The above

tion<sup>[20–23]</sup> in just a few clock cycles. This connects to the biologically-inspired concept of associative memories,<sup>[24–26]</sup> and the powerful notion of associative computing.<sup>[27–32]</sup> Today, the physical implementations of CAMs and Ternary CAM (TCAM)—which introduce a third wild card “x” state besides the usual “0” and “1”—are typically based on SRAM circuits, but have high area and cost.<sup>[33]</sup> Utilizing nonvolatile technology has been explored to reduce power and area, including with memristive devices<sup>[34–39]</sup> and ferroelectric technology.<sup>[40,41]</sup> Taking advantage of the analog tunability and dynamic range in such devices an analog (or multi-bit) CAM based on memristors<sup>[42]</sup> was introduced. The concept is to store ranges in each CAM cell using two memristors that define the boundaries, either in discrete allowed intervals (multi-bit) or continuously tunable (analog). Similarly, the inputs can take discrete values (multi-bit input) or be continuous. This can offer higher data density and lower energy per search,<sup>[42]</sup> but does become susceptible to variability and fluctuations in memristive devices which continue to mature. Additionally, unique operations such as inequality tests can be conducted with an analog CAM. This has been described and utilized to accelerate the decisions in tree-based machine learning models, including random forests.<sup>[43]</sup> The above

applications, however, focused on digital outputs (match or mismatch) from the CAM or analog CAM. Yet, in the CAM circuits, the digital output signal consists of several analog comparisons and operations within the CAM array. For example, a mismatch corresponds to the capacitive charge in the matchline being discharged by current flowing in the CAM cells. This means that within CAMs, more quantitative information about the input/output relationships is present but not used. In particular, by measuring the analog, rather than discrete, output it is possible to elucidate the continuous relationship between output and the inputs or storage. Interestingly, this makes the analog CAM output a scalar field, dependent on multiple parameters including the input, stored values, and even parameters intrinsic to the circuit realization. Furthermore, the gradient of this scalar field can be computed which enables optimization (e.g., gradient descent) and learning techniques to be utilized. The present work studies this novel concept of a differentiable CAM (dCAM), which is shown to act as a soft memory and by sensing analog quantities as outputs, new capabilities are developed. After describing the operational concept we illustrate the new capabilities through several applications, namely i) the capability to improve analog data storage programming values for enhanced accuracy, ii) mitigation of memristor device variability and fluctuations, iii) improved energy and power by choosing circuit parameters, and iv) the solution of  $k$ -SAT optimization problem by directly encoding Boolean clauses and learning the optimal inputs (variable assignment) to satisfy the clauses. These results illustrate the broad range of applications for a memristive differentiable analog CAM, which can be used as a trainable module capable of learning and acting as a part of a larger neural-inspired architecture.

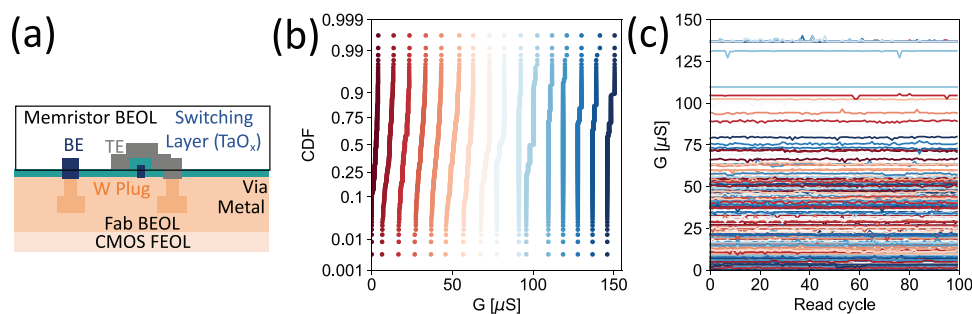
## 2. Memristor Integration and Programming

We developed the dCAM based on the previous analog CAM using memristors.<sup>[42]</sup> Memristors are two terminals devices that act as a nanoscale programmable resistance. **Figure 1a** shows the side view of our BEOL fabricated memristor,<sup>[44,45]</sup> where a  $\text{TaO}_x$  layer is sandwiched between a metallic top electrode (TE) and bottom electrode (BE). By applying a positive voltage on the TE, a device that is initially in a pristine state undergoes a set transition resulting in a low resistance. By controlling the maximum current flowing during the set operation, this resistance can be modulated. For reset, a negative voltage is applied to the TE and by controlling the maximum voltage applied it

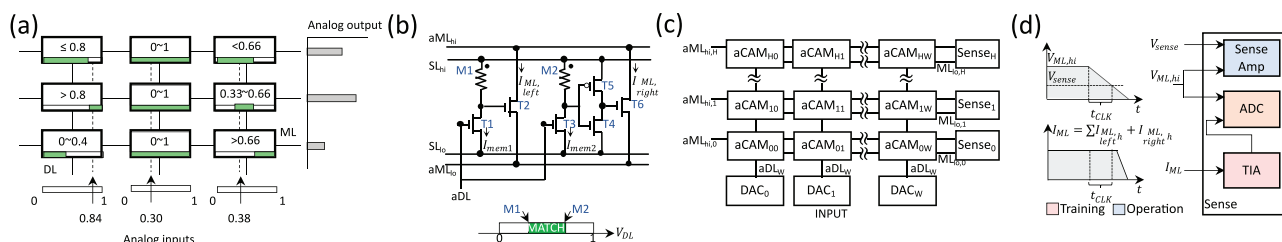
is possible to modulate the final high resistance state.<sup>[46]</sup> Controlling the analog conductance by either set current or reset voltage is not trivial due to stochastic processes present in the resistance switching mechanisms. **Figure 1b** shows the cumulative distribution of 16 equally spaced conductance states programmed in 200 memristor devices, demonstrating both clear levels and device-to-device variations in conductance. Multiple techniques have been proposed to address analog variations,<sup>[47–51]</sup> such as using multiple devices for representing a single entry and thus paying an area and energy consumption overhead. These techniques are more complicated for non-conventional memory circuits, such as CAMs<sup>[37,42]</sup> which require more CMOS circuitry than crossbar arrays and also introduce nonlinear components in the circuit's input–output relationship. Furthermore, following the programming, memristive devices can show stochastic fluctuations (**Figure 1c**) particularly in high resistance states. In CAMs, these fluctuations may translate to errors in the CAM lookup operation over time (e.g., a CAM row producing a false match or false mismatch).

## 3. Differentiable CAM

**Figure 2a** shows the dCAM concept, consisting of a CAM circuit with analog input and analog storage similar to the analog CAM<sup>[42]</sup> but with the additional feature of an analog output on the match line (ML). In conventional CAM operation, the ML is precharged and if the operation returns a match, the ML stays charged, otherwise it discharges. The mismatch is typically obtained due to a current flowing into the CAM cells which discharges the capacitance on the ML. **Figure 2b** shows an analog CAM circuit<sup>[42]</sup> (a layout in 16 nm CMOS is shown in **Figure S1**, Supporting Information). The analog input voltage  $aDL$  is applied to the gate of two different transistors, namely T1 and T3. T1 is responsible for dividing the  $SL = SL_{hi} - SL_{lo}$  voltage with the memristor M1. When the T1 conductance is low compared to M1 (i.e., low  $aDL$ ), the voltage at the middle point between T1 and M1 will be high, and thus T2 will activate and drain a large current  $I_{ML, left}$  from the match line  $aML_{hi}$ , eventually discharging ML (a mismatch). When the T1 conductance is large compared to M1 (i.e., high  $aDL$ ), T2 will not activate and this results in a match. Thus the left branch of the circuit composed of T1, M1, and T2 verifies if the input is larger than the stored value of the left boundary encoded by M1. The right branch operation is similar, with the exception that



**Figure 1.** a) Side-view of material stack for BEOL integrated  $\text{TaO}_x$  memristors. The CMOS circuit is fabricated by an external Fab in the front end of the line (FEOL) which the memristors are integrated in our clean room in the back end of the line (BEOL). b) Distribution of conductance for 16 equally spaced levels programmed in 200 memristors each. c) Read noise of 200 conductance states as a function of read cycle.



**Figure 2.** a) dCAM concept, where analog input are searched and a similarity between input and the analog stored value is returned as output. b) Circuit schematic of dCAM (inset shows the range encoding) and c) HxW array organization. d) Analog quantities, that is,  $V_{MLhi}$  and  $I_{ML}$  are sensed by the training circuits (ADC and TIA+ADC) and the operating circuit (sense amplifier only) used after training completes.

a CMOS inverter (T4–T5) is added between the voltage divider (M2–T3) and pull-down transistor T6 which is responsible for discharging the  $aML_{hi}$ . A high aDL activates T6, corresponding to a mismatch operation, thus this right branch is responsible for sensing if the input is smaller than the stored value of the right boundary encoded by M2. Overall, an analog CAM cell circuit verifies if an input is within the range stored in the cell.

To operate as dCAM the circuit has been modified, adding the capability to sense the discharge current on  $aML_{lo}$ , representing a distance metric of the input compared with the stored data in the row. If the input is close to the stored values, a small current is expected to flow in  $aML_{lo}$  and if the input is significantly different from the stored values, the current increases and discharges  $aML_{hi}$  quickly. During dCAM operation, the  $aML_{hi}$  can be either kept at fixed value or precharged. In the former case, the current is constant and depends on the voltage on aDL and the stored conductance. In the latter case, the current changes while the  $aML_{hi}$  discharges but to first approximation we assume constant while the T2 and T6 are in saturation regime. Figure 2c shows a typical array organization of a CAM, where an input is provided at the columns and the state of each row ML would normally correspond to a bitwise AND operation across the individual cells of that row with the input. The array now includes the needed dCAM capabilities, including analog inputs which are supplied in this case through digital to analog converters (DAC) on the columns, and analog outputs which can be sensed either on the  $aML_{hi}$  or  $aML_{lo}$ . Figure 2d illustrates the sensing circuits used for enabling dCAM operation, namely an analog to digital converter (ADC) for sensing the voltage on  $aML_{hi}$  and a transimpedance amplifier (TIA) to convert the current flowing in  $aML_{lo}$  into a voltage sensed with an analog to digital converter (ADC). In the dCAM training mode, analog inputs are applied and by sensing the analog outputs and computing the gradients it is possible to optimize many parameters (such as storage values i.e., the memristor conductance, or inputs) for different tasks. After training, depending on the application, the dCAM can be operated either by using the analog output directly, or as an analog CAM converting the

voltage on  $aML_{hi}$  into a binary match/mismatch signal with sense amplifier after a given clock time  $t_{CLK}$ . Table 1 shows a direct comparison of different types of CAM concepts.

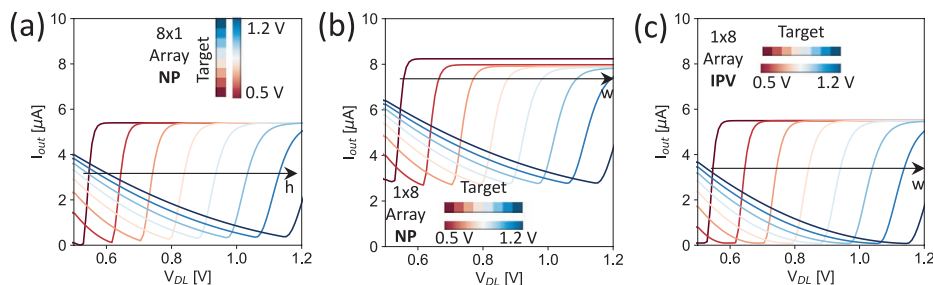
## 4. Results and Discussion

### 4.1. Improved CAM Programming: Learn to Store with a Multi-Bit dCAM

In conventional crossbars, choosing the desired conductance to program the memristive devices for ideal matrix vector multiplication operation<sup>[18]</sup> is straightforward as the operation is linear (e.g., if a current  $I$  is desired when the input is  $V$ , the target conductance should be  $G = I/V$ ). This is not the case for analog CAMs as the transfer function is inherently nonlinear as several voltage/current transformations through transistors are involved. By knowing the cell design, it is possible to estimate the required memristor conductance to program into an analog CAM cell to match a given range, that is, keep the ML charged for at least one clock cycle  $t_{CLK}$  and discharge it otherwise, as described in Algorithm S1, Supporting Information and referred here as the Naïve programming (NP) approach (circuit variable description in Table S1, Supporting Information). The current  $I_{ML}$  required for discharging  $aML_{high}$  in  $t_{CLK}$  (mismatch), is first computed and then reported as the required conductance of M1 and M2. We used the NP approach for programming eight levels (or  $N_{bit} = 3$ ) into an analog CAM. Figure 3a shows the simulated current  $I_{ML}$  as a function of input voltage  $V_{DL}$  for each analog CAM cell (different colors) as a result of programming an equally spaced  $V_{DL}$  target between 0.5 and 1.1V (inset) in eight analog CAMs arranged in a column, an  $8 \times 1$  array. A tolerance of  $\tau = 50\text{mV}$  was added, namely a CAM should match if the input  $V_{target} - \tau < V_{DL} < V_{target} + \tau$ . In this case, the single input on the  $V_{DL}$  is sensed separately by each cell and the current  $I_{ML}$  of each  $aML_{hi}$  is minimized when the input corresponds to the target. Note that the simulation disregards the nonlinearity of the memristor conduction given its low impact

**Table 1.** Comparison between CAM concepts.

Circuit	Input	Storage	Output
TCAM [37]	Digital (ternary)	Digital (ternary)	Digital
Analog CAM [42]	Analog	Analog	Digital
Differentiable CAM [this work]	Analog	Analog	Analog



**Figure 3.** a) Programming equally spaced  $V_{DL}$  pattern (inset) in an  $8 \times 1$  analog CAM array with Naïve programming (NP). Each line represents the input–output relationship for different cells in the  $8 \times 1$  array, where color is the target voltage programmed in it. b) Same pattern programmed with NP in a  $1 \times 8$  array showing an undesired current offset. c) Same pattern correctly programmed in a  $1 \times 8$  array with CAM aware iterative program and verify (IPV) which significantly reduces the current.

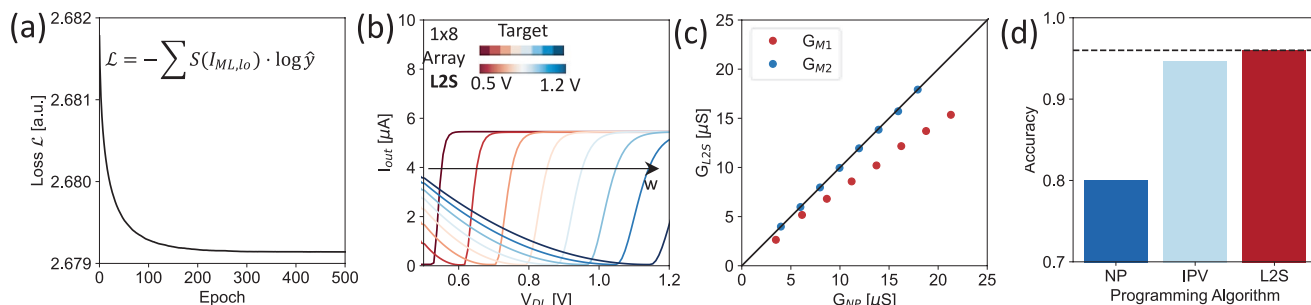
in our devices.<sup>[42]</sup> While this programming algorithm is effective, it does not completely cancel the currents corresponding to target inputs, in order to avoid superposition between levels, and a small leakage is still present. For example when giving the highest target, that is,  $V_{DL} = 1.1$ , a leakage current of  $I_{leak} = 0.4 \mu A$  is flowing from  $ML_{hi}$  to  $ML_{low}$ . This can introduce errors when considering the same target programmed in a  $1 \times 8$  CAM array, or row vector. Figure 3b shows  $I_{ML}$  as a function of  $V_{DL}$  for the same pattern of Figure 3a programmed in a row, instead of a column. Each cell was tested by keeping all the other to a perfect matched (i.e., target) input and sweeping the tested CAM input voltage. As the figure shows, a large offset is present due to all small currents contributions of each matched cell. To circumvent this problem, we developed a CAM-aware iterative program and verify algorithm (IPV)<sup>[43]</sup> as described in Supplementary Algorithm S3, Supporting Information. IPV starts by programming the row with the NP (see Algorithm S2, Supporting Information), then the row is tested with the target input and if it results in a mismatch, NP is repeated again with more stringent match/mismatch requirements. To do that, the sense amplifier threshold voltage  $V_{sense}$  is slightly increased during the calculation of the required  $G_{M1}$  and  $G_{M2}$ . The process is repeated until convergence, that is, until a match is returned after testing with the input target vector. Figure 3c shows the same pattern of (a) and (b) programmed in an  $1 \times 8$  array efficiently with IPV.

IPV is a solution for programming patterns efficiently in analog CAM, however it is not resilient to device/circuit variations and noise since knowledge about such issues is not included in the programming operation. Moreover, an optimal  $\delta V$  is difficult to find to achieve a good balance between number of iterations and convergence. For this reason, with dCAM we developed a Learn to Store (L2S) procedure, which exploits the analog in/analog storage/analog output capabilities to learn the target vector to store in the memory circuit. Previously, we developed a compact analog CAM model<sup>[43]</sup> to quickly assess the input/output relationship of large models without performing SPICE circuit simulations. Here, we translated the models to PyTorch<sup>[52]</sup> including the dCAM analog capabilities, in order to enable auto-differentiation. L2S can be seen as solving a classification problem, where the goal is to learn how to output a match in a given dCAM row when applying the input target vector. L2S can be performed either in software simulations or in hardware, but in both case an accurate model of the hardware is needed for computing the gradients.

As an initial test for L2S, we reproduce the results of programming with the IPV algorithm, but emerging through supervised training instead. We consider at this stage only ideal devices (i.e., without noise and variability issues for memristors and CMOS circuitry). In order to learn the target vector to store of Figure 3c, we treat this as a classification problem, where the goal is to program the desired pattern such that it correctly accepts the set of correct input patterns, and rejects the set of incorrect input patterns. Our study here is performed in simulations where the output analog signals are computed with our experimentally-grounded circuit model. In a hardware implementation of L2S, the row output would be directly measured in the hardware shown in Figure 2c. For a given target vector, a dataset for learning is created. The training dataset is generated by combining the input pattern (inset of Figure 4a) that should be accepted (labeled as match), with the patterns to reject (mismatch) that consist of 1 bit flips of neighboring inputs in the multi-bit problem. The training dataset is augmented by extracting 100 random analog input values that fall within each pattern range (see Figure S2, Supporting Information). The dCAM is initialized with the conductance obtained by programming with NP. The dataset is then applied to the dCAM and conductance learning is performed by optimizing the cross-entropy loss of the SoftMax of  $I_{ML,lo}$  for each row in a CAM, where the target  $\hat{y}$  represents the memory address, that is, row number for where the target pattern will be within the CAM array. Figure 4a, shows the loss as a function of learning epoch demonstrating correct minimization. In fact the dCAM input/output relationship after training in Figure 4b is comparable with the result obtained with IPV in Figure 3c. Note this result is expected since IPV is already iteratively adjusting the conductance based on the circuit answer, which is a simplified version of what L2S does in a more formal way. Figure 4c shows the final conductance obtained with L2S as a function of the initial conductance (i.e., obtained with NP), where  $G_{M1}$  are reduced to avoid leakage currents and hence false mismatches. The shift has a nonlinear dependence on the conductance state, as shown in the figure. While the resulting behavior is intuitive and matches our intelligent IPV algorithm, it is notable that it was not explicitly programmed here, but emerged from the learning procedure.

The effectiveness of L2S is further quantified by using it in a CAM based acceleration task. We recently developed a tree based ML inference accelerator that utilizes the novel inequality testing possible in an analog CAM.<sup>[43]</sup> Each root to leaf





**Figure 4.** a) Training Loss during learning as a function of epochs. The loss corresponds to the cross-entropy of  $I_{ML}$  SoftMax  $S$  function (see inset equation). The dataset was created by inserting dummy rows for each 1-bit mismatch. b) Pattern of Figure 3 learned by the dCAM with L2S replicating the same result of IPV. c) Final conductance after L2S as a function of initial conductance, namely the conductance obtained with NP. d) DT inference accuracy comparison for three programming strategies with digital software accuracy in dashed line.

path (branch) of a decision tree (DT) is mapped to an analog CAM array row, as shown in Figure S3, Supporting Information, and thus allows single-step tree inference. The accuracy of the inference operations is critically dependent in faithfully programming the stored CAM words that encode these DT inequality values. We assessed the capabilities of L2S in dCAM for this application, using the cross-entropy loss of the SoftMax of the voltage  $V_{ML,hi}$  after a clock cycle  $t_{CLK}$ . This corresponds to a training of the dCAM to efficiently perform DT inference (see Figure S4a, Supporting Information). Note that this is a different operation compared to the above multi-bit storage application, since during inference the sense amplifier is used. First, we trained a DT model for classification of the Iris Dataset<sup>[53]</sup> with conventional tools (i.e., Python Sk-Learn) offline. After we extracted the trained DT thresholds, we utilize dCAM to learn how to store them in the optimal way with L2S. Supplementary Figure 4b shows the loss as a function of the training epochs demonstrating correctly minimization. Finally, we compared L2S with other non-gradient based algorithms as shown in Figure 4d. The accuracy in DT inference for the three algorithms, namely NP, IPV, and L2S is plotted, demonstrating that only L2S performs inference with equivalent software accuracy (dashed line). Note that this result was obtained with ideal components, thus noise-less memristor and defect/parasitic free CMOS circuitry. By inserting in the compact model knowledge about the variability, noise and parasitics present in the circuit we next show it is possible to compensate for such issues.

## 4.2. Memristor Variability Compensation

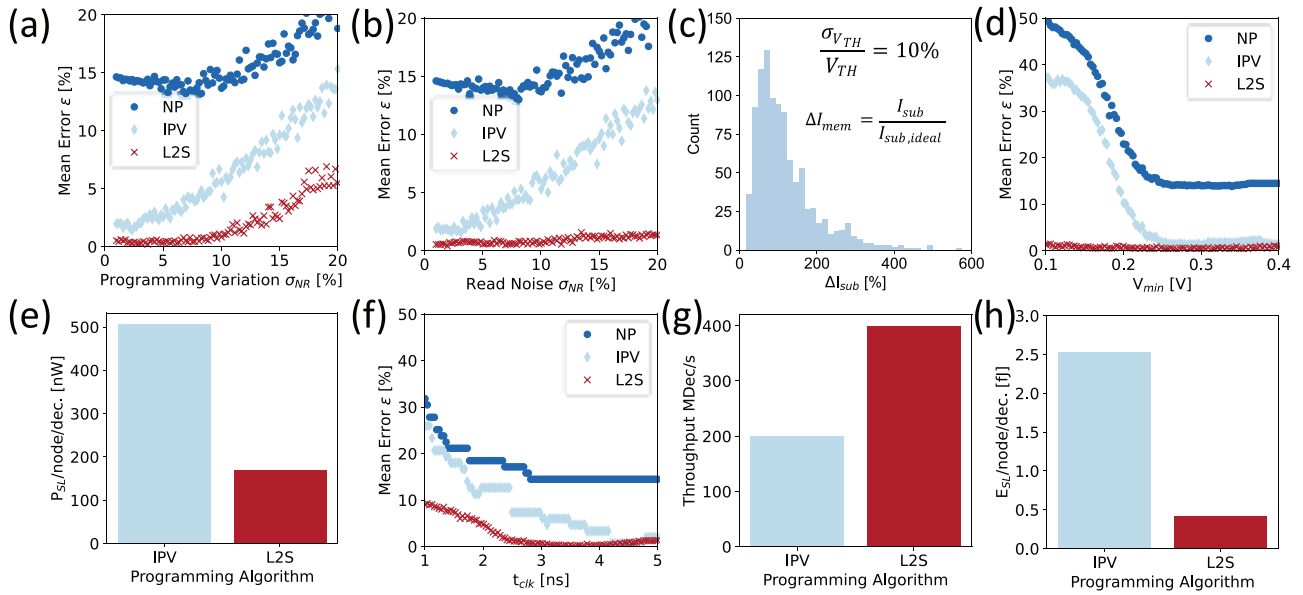
Memristor devices can be programmed to multiple levels or conductance states. However, as the programming operation contains stochastic processes,<sup>[18]</sup> the programmed conductance shows variability, as previously seen in Figure 1. By embedding a stochastic conductance with a constant standard deviation  $\sigma_{NR}$  for each level in the dCAM during training, it is possible to program the conductance in an optimal way regardless of variation. Note that this procedure is similar to the in situ training operation for conventional crosspoint arrays.<sup>[8]</sup> Figure 5a shows the error during inference  $\varepsilon = A_{SW} - A_{HW}$ , with  $A_{SW}$  ideal software accuracy and  $A_{HW}$  the inference accuracy after deploying

the model on analog CAM or dCAM, as a function of the programming variation  $\sigma_{NR}$ . The calculation was repeated for the three programming strategies of NP and IPV for analog CAM and L2S for dCAM. While analog CAM error increases even for small  $\sigma_{NR}$ , dCAM is able to compensate the noise up to  $\approx 10\%$  without loss in accuracy due to the use of a training algorithm. Memristors are also subject to read noise, meaning that the conductance also varies through time.<sup>[18]</sup> We considered random conductance fluctuations in time with a normal distribution across the programmed value and similarly assessed the resulting error during inference for the different programming approaches. Figure 5b shows the  $\varepsilon$  as a function of the read noise standard deviation  $\sigma_{NR}$  demonstrating that dCAM maintains an almost constant accuracy even for large  $\sigma_{NR}$ .

## 4.3. dCAM for Optimizing Circuit Performance

### 4.3.1. Reducing Power by CMOS Sub-Threshold Current Variability Compensation

An analog CAM can have large static power consumption due to current flowing from  $SL_{hi}$  to  $SL_{lo}$  in the voltage divider operation.<sup>[43]</sup> Lower power analog CAM operations are possible by operating the input transistors T1 and T3 in the subthreshold regime, limiting static current and thus static power. Another benefit to this operational regime is that the smaller  $V_{DL}$  corresponds to adopting smaller conductance as stored thresholds, which is also beneficial for total search power. Unfortunately, subthreshold conduction of transistors suffers from large variability due to the exponential dependence of the transistor threshold voltage  $V_{TH}$  which is subject to process variation.<sup>[54]</sup> Figure 5c shows the probability distribution of the subthreshold transistor current error, namely  $\Delta I_{mem} = I_{sub}/I_{sub,ideal}$  where  $I_{sub}$  is the current affected by process variation and  $I_{sub,ideal}$  the nominal one, for a threshold variation  $\sigma_{V_{TH}}/V_{TH} = 10\%$ ,<sup>[54]</sup> where a large variation is evident. By inserting information about the process defect in the dCAM model, we tested the ability to learn high inference accuracy while simultaneously reducing the minimum voltage  $V_{min}$  applied to  $V_{DL}$  to minimize power. Figure 5d shows the inference error  $\varepsilon$  as a function of  $V_{min}$  demonstrating that the learning procedure in



**Figure 5.** Efficient inference of DT model with analog CAM. a) Inference error  $\varepsilon = A_{SW} - A_{HW}$  with  $A_{SW}$  software baseline and  $A_{HW}$  hardware accuracy, as a function of programming variability standard deviation for the three programming strategies. b) Inference error  $\varepsilon$  as a function of read noise standard deviation for the three strategies. c) Distribution of subthreshold transistor current error for a 10% standard deviation of the error on the threshold. d) Inference error  $\varepsilon$  as a function of minimum applied  $V_{DL}$  for the three programming strategies. e) Iso-accuracy SL static power consumption per node for IPV and L2S. The latter can operate at lower  $V_{min}$ . f) Inference error  $\varepsilon$  as a function of clock time  $t_{CLK}$  for sensing  $ML_{hi}$ . g) Resulting throughput for the different programming strategies. h) Total resulting energy consumption on the SL path.

the dCAM is robust to process variation while analog CAM implementation both with NP and IPV strategies are strongly affected. The result is that the dCAM enables lower power consumption for CAM applications. By considering an inference iso-accuracy with analog CAM, Figure 5e shows the energy per node per decision<sup>[37,43]</sup> which is reduced by a factor 3× with L2S and dCAM compared with IPV and analog CAM.

#### 4.3.2. Higher Throughput and Lower Energy by Parasitic Capacitance Compensation

Finally, we test dCAM in compensating the parasitic capacitance which is formed on the  $ML_{hi}$  for each cell that is added in a row. The overall capacitance can be computed as  $C_{ML} = C_{PC} + C_{sense} + WC_{CAM}$  where  $C_{PC}$  is the output capacitance of the precharge circuit,  $C_{sense}$  is the input capacitance of the sense amplifier circuit,  $W$  is the total number of CAM cells in a row and  $C_{CAM}$  the parasitic capacitance which correspond to the drain-source capacitance of transistor T2 and T6 connected to the  $aML_{hi}$ .<sup>[43]</sup> Note that during NP/IPV the parasitic capacitance is considered, but without the collective behavior of discharge currents. By embodying this information in the dCAM model, it is possible to learn efficiently the proper conductance to program in order to maximize the clock speed as shown in Figure 5f. Smaller clocks corresponds to larger throughput, namely the number of decision per second that can be inferred by the CAM. Figure 5g shows the throughput for analog CAM with IPV and dCAM with L2S which is 2× larger for the latter. Note that after L2S dCAM is operated as an analog CAM thus does not require analog to digital conversion of the output, thus the high throughput. As a result, by storing a model with L2S

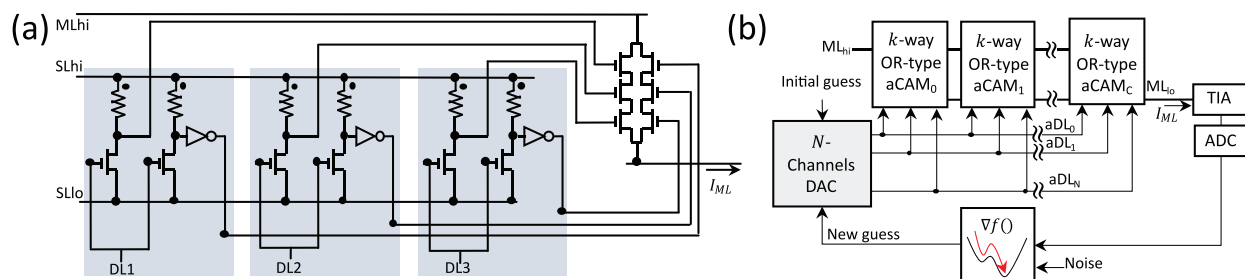
optimizing both for process variation and parasitic mitigation it is possible to reduce the energy consumption of the SL by a factor 6× as shown in the plot of Figure 5h. This is due to the joint effects of static power minimization by using subthreshold input transistors and parasitic capacitance compensation.

#### 4.4. Solving $k$ -SAT Optimization Problems

The dCAM enables entirely new functionality not typically tackled with CAM circuits. In particular, we studied the application to Boolean satisfiability ( $k$ -SAT), a class of NP-complete problems,<sup>[55]</sup> to which many other optimization problems can be mapped.<sup>[56]</sup> An example of a  $k$ -SAT (with  $k = 3$ ) problem is finding the binary variable assignments that return as output “1” of a given conjunctive normal form of equations. As an example, given the Boolean expression

$$(x \vee \neg y \vee z) \wedge (\neg x \vee \neg y \vee \neg z) \quad (1)$$

the variable assignment [1,0,0] to the inputs satisfies the expression. Interestingly, OR-type CAMs can act as  $k$ -SAT filter, namely they can be used to verify if an input set of variables satisfies the Boolean expression.<sup>[57]</sup> We extended this concept to the dCAM in order to learn the proper set of inputs which solves the encoded problem. Figure 6a shows the needed modifications to the CAM cell to perform a three-way OR operation. Each CAM cell returns a match if at least one of the three input literals is within the range encoded by the memristors. Given the clauses of Equation (1), they can be mapped as “1” or “0” in the 3-way OR type CAM. If a literal is negated a “1” should be programmed, that is, the lower branch memristor should be programmed to a



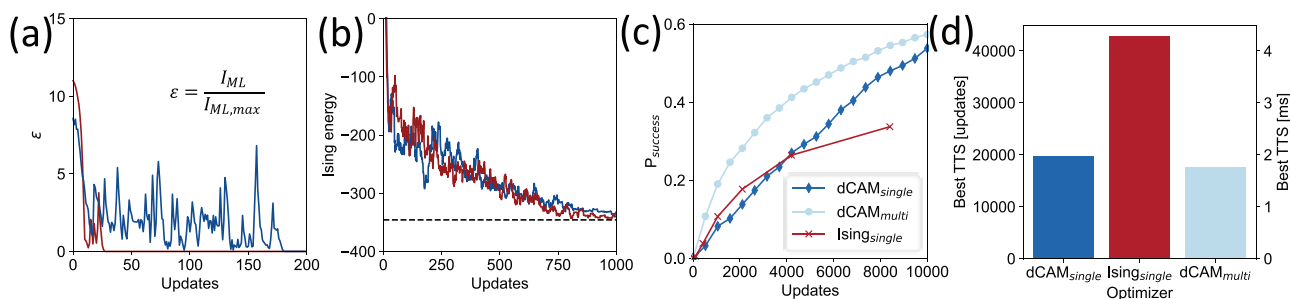
**Figure 6.** a) Three-way OR-type dCAM which results in a “Match” if any of the three DL inputs satisfies the memristor-encoded logical expression b)  $k$ -SAT solution with dCAM. Given a problem with  $N$  variables,  $C$  clauses and  $k$  literals this can be encoded in  $C$   $k$ -way analog CAM cells whose output is differentiated and given to an  $N$ -channel DAC driving the aDL. The programmed conductances encode the problem, while the correct input is learned. Noise is added to escape local minima (i.e.,  $I_{ML} > 0$  for multiple iterations) in the solution space. Different techniques such as simulated annealing or basin hopping can be applied. When  $I_{ML} \approx 0$  the problem is solved.

reasonable middle state, for example,  $G = 50 \mu\text{S}$  and the upper branch memristor to the state corresponding to a do not care, or always match, which in this case is a low conductance state. The opposite mapping would apply to a literal which is not negated, thus the lower branch memristor should be programmed to a high conductance state and the upper branch memristor to a reasonable middle state. If the proper set of input, for example,  $[1,0,0]$ , is applied to the transistors connected to the  $ML_{hi}$  (performing the OR operation) remain off, otherwise they turn on and the current  $I_{ML}$  is a measure of how distant the candidate input is from the correct solution. This analog information can be differentiated and gradient-based searching or annealing algorithms used to find the optimum set of inputs that minimizes the output. Note that while memristor values do not change during the solution of  $k$ -SAT, it is required to use them instead of simple resistors in order to program different Boolean expressions when encoding a new problem into the hardware. Figure 6b shows a circuit for solving  $k$ -SAT problems with the OR-type dCAM concept. Given a problem with  $N$  variables,  $C$  clauses and  $k$  literals a  $k$ -way OR type dCAM should be used. The inputs are applied with an  $N$  channels DAC and the resulting output current  $I_{ML}$  is read with a TIA and ADC chain and fed into an optimizer.

#### 4.4.1. Comparison with Ising Solvers

We tested the dCAM with 100 three-SAT problems taken from the SATLIB-uf20.91 dataset,<sup>[58]</sup> consisting of instances with  $N = 20$  variable and  $C = 91$  clauses, which is considered to be

in the spectrum of hard problems because the ratio of clauses to variables  $C/N \approx 4.5$ .<sup>[59]</sup> We compared dCAM with the popular Ising approach, which maps 3-SAT problems to an Ising Hamiltonian by first mapping it to a maximum independent set (MIS) problem, and finds the ground states.<sup>[60]</sup> As a hardware approach for comparison, we considered a memristive Hopfield neural network<sup>[12]</sup> which can encode Ising problems and solve at high throughput. But even this highly efficient memristor solver requires a total of  $(kC)^2$  memristors, resulting in poor scaling. Interestingly, the dCAM solver only needs  $2kC$  memristors, scaling linearly with the problem size. Figure 7a shows the dCAM error  $\varepsilon = I_{ML}/I_{ML, \max}$ , with  $I_{ML, \max}$  the saturation current of the transistors as a function of update cycles. Noise is added externally to the solver and different stochastic approaches such as simulated annealing and/or basin hopping can be performed. This can be done either by software tools or directly in hardware.<sup>[12,14]</sup> Figure 7b shows the Ising energy for the same problems as a function of the update cycles, which requires a larger number of cycles to converge. Figure 7c shows the probability to solve a problem as a function of the update cycle for dCAM (dCAM<sub>single</sub>) and an Ising model (Ising<sub>single</sub>) demonstrating the better performance of dCAM up to  $1.5\times$ . dCAM was also tested with all the problems to compute the average probability of solving a problem (dCAM<sub>multi</sub> in Figure 7c) which is in good agreement with the problem chosen. Figure 7d shows the time to solution (TTS),<sup>[12]</sup> namely the time required to have a 99% probability of finding the right solution for the two implementation, including an average over 100 problems for dCAM accelerator. The results show that dCAM has a



**Figure 7.** Example runs showing error  $\varepsilon$  for a) dCAM optimizer and b) Ising energy for an Ising solver as a function of updates for two different problem initializations. Comparison of probability to correctly solve a c) three-SAT problem and best time to solution in number of updates (left axis) and d) milliseconds (right axis) for a 99% probability of success for dCAM optimizer (dCAM<sub>single</sub>) and Ising solver (Ising<sub>single</sub>). dCAM optimizer was also tested (dCAM<sub>multi</sub>) with 100 different three-SAT problems to show the average behavior.

significant advantage of  $2\times$  faster compared with Ising solver, in addition to reduced requirements and better scaling in the number of memristive elements. It is thus anticipated that the performance improvement would be even more striking for larger problem sizes. Interestingly, our result is close to running miniSAT algorithm, among the top SAT solvers, in an FPGA<sup>[61]</sup>, which takes  $\approx 0.8$  ms and consuming  $\approx 3$  W. Based on previous results<sup>[42,43]</sup> we expect our solver to have a peak power consumption of  $\approx 5$  mW, suggesting at least two orders of magnitude lower energy to solution. With this promising performance indication, future studies will be needed to expand this comparison to include more problem classes, variable sizes, benchmark instances, and hardware approaches.

## 5. Conclusion

Here we demonstrated the novel concept of a differentiable analog nonvolatile CAM. This new concept enables the ability to learn what patterns to store to improve overall accuracy, depending on the desired application. This was demonstrated in the programming approach for tree based ML acceleration in CAMs to compensate for device and circuit nonidealities. Beyond setting the patterns to store and improving robustness to variability, gradients of the output can be taken with respect to circuit parameters in order to yield performance improvements such as lower power, energy, or increased throughput. Finally, the concept was extended to solving optimization problems, where the constraints are mapped in the CAM and the dCAM learns the optimal set of inputs to satisfy Boolean expressions. We envision the use of dCAM as a core building block of fully differentiable computing system employing multiple types of analog compute operations and memories, and thus broadening the set of application even further than those shown here.

## 6. Experimental Section

**CMOS and Memristor Integration Process:** The memristors were monolithically integrated on CMOS fabricated in a 180 nm technology node. The integration starts with a removal of silicon nitride and oxide passivation with reactive ion etching and a buffered oxide etch dip. Chromium and platinum bottom electrodes were then patterned with e-beam lithography and metal lift-off process, followed by reactive sputtered 4.5 nm tantalum oxide as switching layer, with 10% of oxygen in a Ar:O<sub>2</sub> mixture. The BE metal was thus Cr/Pt, with Cr as an adhesion layer. The device stack was finalized by e-beam lithography patterning of sputtered tantalum and platinum metal as top electrodes. The TE was then Ta/Pt/Cr, with Pt a protection for oxidation of Ta and Cr as a etch mask to etch Pt and Ta. The effective electrodes for the memristors (making direct contact with TaO<sub>x</sub>) were Pt for BE and Ta for TE. More information about the fabrication process and memristor characterization can be found in refs. [44,45].

**Memristor Characterization:** Memristor devices were characterized in a fully integrated system consisting in three  $64 \times 64$  arrays of memristors in 1-transistor-1-memristor configuration, routing, and sensing circuits.<sup>[45]</sup> The system was taped out in 180 nm CMOS technology and memristors integrated in Hewlett Packard Labs clean room facilities. Program and verify algorithm were used for programming the desired conductance by an increasing either the gate or top electrode voltage during programming. First, a read operation was performed to each

device. If a memristor conductance  $G_{ij}$  was bigger than its target  $G_{\text{target},ij}$ , namely  $G_{ij} > G_{\text{target},ij} + g_{\text{tol},\text{in}}$  where  $g_{\text{tol},\text{in}}$  is the internal tolerance, a reset operation was performed by increasing at each step the BE voltage from 0.5 to 1.5 V with steps of 0.1 V and increasing the gate voltage from 0.5 to 3 V. In contrast, if  $G_{ij} < G_{\text{target},ij} - g_{\text{tol},\text{in}}$  a set operation is performed by increasing at each step the TE voltage from 0.5 to 2 V with steps of 0.1 V and increasing the gate voltage from 0.5 to 1.5 V. The programming operation was repeated until convergence, namely until  $|G_{ij} - G_{\text{target},ij}| < g_{\text{tol},\text{in}}$ . While programming a given memristor, it was possible that another memristor in another location changes its conductance even if its programming operation already converged, due to both device and circuit nonidealities. For this reason, if  $|G_{ij} - G_{\text{target},ij}| > g_{\text{tol},\text{out}}$ , where  $g_{\text{tol},\text{out}} > g_{\text{tol},\text{in}}$  is the external tolerance, the programming operation is repeated. More details about the program and verify algorithm can be found in ref. [62].

**dCAM Modeling:** dCAM was modeled in PyTorch environment as a module which includes the current–voltage transfer function of transistor. Memristor read and program noise was modeled as a Gaussian distribution with constant  $\sigma_C/G$ . For read noise, the samples were taken from the distribution at each iteration while for programming variation samples were taken only when the memristor conductance was updated. Process variations were modeled as  $V_T$  initialization by sampling before training from a Gaussian distribution. Fitting values for the model and parasitic were extracted from the taped-out chip at 180 nm technology<sup>[42]</sup> and post layout simulation at 16 nm technology.<sup>[42,43]</sup>

## Supporting Information

Supporting Information is available from the Wiley Online Library or from the author.

## Conflict of Interest

The authors declare no conflict of interest.

## Data Availability Statement

The data and code that support the findings of this study are available from the corresponding author upon reasonable request.

## Keywords

analog computing, content addressable memories, in-memory computing, memristor

Received: November 5, 2021

Revised: January 17, 2022

Published online:

- [1] D. Ielmini, H.-S. P. Wong, *Nat. Electron.* **2018**, 1, 333.
- [2] J. Hasler, in *2016 IEEE Int. Conf. on Rebooting Computing (ICRC)*, IEEE, Piscataway, NJ **2016**, pp. 1–8.
- [3] M. A. Zidan, J. P. Strachan, W. D. Lu, *Nat. Electron.* **2018**, 1, 22.
- [4] 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. N. Kurdi, H. Hwang, *IEEE Trans. Electron Devices* **2015**, 62, 3498.
- [5] A. Shafiee, A. Nag, N. Muralimanohar, R. Balasubramanian, J. P. Strachan, M. Hu, R. S. Williams, V. Srikumar, in *2016 ACM/IEEE 43rd Annual Int. Symp. on Computer Architecture (ISCA)*, IEEE, Piscataway, NJ **2016**, pp. 14–26.



- [6] S. Agarwal, S. J. Plimpton, D. R. Hughart, A. H. Hsia, I. Richter, J. A. Cox, C. D. James, M. J. Marinella, *International Joint Conference on Neural Networks* **2016**, 10.
- [7] S. Ambrogio, P. Narayanan, H. Tsai, R. M. Shelby, I. Boybat, C. di Nolfo, S. Sidler, M. Giordano, M. Bodini, N. C. P. Farinha, B. Killeen, C. Cheng, Y. Jaoudi, G. W. Burr, *Nature* **2018**, 558, 60.
- [8] C. Li, D. Belkin, Y. Li, P. Yan, M. Hu, N. Ge, H. Jiang, E. Montgomery, P. Lin, Z. Wang, W. Song, J. P. Strachan, M. Barnell, Q. Wu, R. S. Williams, J. J. Yang, Q. Xia, *Nat. Commun.* **2018**, 9, 2385.
- [9] V. Joshi, M. Le Gallo, S. Haefeli, I. Boybat, S. R. Nandakumar, C. Piveteau, M. Dazzi, B. Rajendran, A. Sebastian, E. Eleftheriou, *Nat. Commun.* **2020**, 11, 2473.
- [10] P. M. Sheridan, F. Cai, C. Du, W. Ma, Z. Zhang, W. D. Lu, *Nat. Nanotechnol.* **2017**, 12, 784.
- [11] C. Li, M. Hu, Y. Li, H. Jiang, N. Ge, E. Montgomery, J. Zhang, W. Song, N. Dávila, C. E. Graves, Z. Li, J. P. Strachan, P. Lin, Z. Wang, M. Barnell, Q. Wu, R. S. Williams, J. J. Yang, Q. Xia, *Nat. Electron.* **2018**, 1, 52.
- [12] F. Cai, S. Kumar, T. Van Vaerenbergh, X. Sheng, R. Liu, C. Li, Z. Liu, M. Foltin, S. Yu, Q. Xia, J. J. Yang, R. Beausoleil, W. D. Lu, J. P. Strachan, *Nat. Electron.* **2020**, 3, 409.
- [13] G. Pedretti, P. Mannocci, S. Hashemkhani, V. Milo, O. Melnic, E. Chicca, D. Ielmini, *IEEE J. Explor. Solid-State Comput. Devices Circuits* **2020**, 6, 89.
- [14] M. R. Mahmoodi, M. Prezioso, D. B. Strukov, *Nat. Commun.* **2019**, 10, 5113.
- [15] M. A. Zidan, Y. Jeong, J. Lee, B. Chen, S. Huang, M. J. Kushner, W. D. Lu, *Nat. Electron.* **2018**, 1, 411.
- [16] Z. Sun, G. Pedretti, E. Ambrosi, A. Bricalli, W. Wang, D. Ielmini, *Proc. Natl. Acad. Sci. U. S. A.* **2019**, 116, 4123.
- [17] J. J. Yang, D. B. Strukov, D. R. Stewart, *Nat. Nanotechnol.* **2013**, 8, 13.
- [18] D. Ielmini, G. Pedretti, *Adv. Intell. Syst.* **2020**, 2, 2000040.
- [19] K. Pagiamtzis, A. Shekholeslami, *IEEE J. Solid-State Circuits* **2006**, 41, 712.
- [20] C. R. Meiners, J. Patel, E. Norige, E. Torng, A. X. Liu, in *Proc. of the 19th USENIX Conf. on Security*, IEEE, Piscataway, NJ **2010**, pp. 1–16.
- [21] K. Peng, S. Tang, M. Chen, Q. Dong, in *2011 ACM/IEEE Seventh Symp. on Architectures for Networking and Communications Systems*, IEEE, Piscataway, NJ **2011**, pp. 24–35.
- [22] F. Yu, R. Katz, T. Lakshman, in *Proc. of the 12th IEEE Int. Conf. on Network Protocols*, IEEE, Piscataway, NJ **2004**, pp. 174–183.
- [23] K. Huang, L. Ding, G. Xie, D. Zhang, A. Liu, S. Kavé, in *Architectures for Networking and Communications Systems*, IEEE, Piscataway, NJ **2013**, pp. 83–93.
- [24] D. J. Willshaw, O. P. Buneman, H. C. Longuet-Higgins, *Nature* **1969**, 222, 960.
- [25] T. Kohonen, *IEEE Trans. Comput.* **1972**, C-21, 353.
- [26] J. J. Hopfield, *Proc. Natl. Acad. Sci. USA* **1982**, 79, 8 2554.
- [27] Q. Guo, X. Guo, R. Patel, E. Ipek, E. G. Friedman, in *Proc. of the 40th Annu. Int. Symp. on Computer Architecture, ISCA '13*, Association for Computing Machinery, New York, NY.
- [28] M. Sharad, D. Fan, K. Roy, in *Proc. of the 50th Annual Design Automation Conf*, Association for Computing Machinery, New York, NY **2013**, pp. 1–6.
- [29] M. Imani, A. Rahimi, D. Kong, T. Rosing, J. M. Rabaey, in *2017 IEEE Int. Symp. on High Performance Computer Architecture (HPCA)*, IEEE, Piscataway, NJ **2017**, pp. 445–456.
- [30] A. Ranjan, S. Jain, J. R. Stevens, D. Das, B. Kaul, A. Raghunathan, in *Proc. of the 56th Annual Design Automation Conf. 2019, DAC '19*, Association for Computing Machinery, New York, NY **2019**.
- [31] Y. Zha, J. Li, in *2020 ACM/IEEE 47th Annual Int. Symp. on Computer Architecture (ISCA)*, IEEE, Piscataway, NJ **2020**, pp. 846–859.
- [32] D. Kleyko, M. Davies, E. P. Frady, P. Kanerva, S. J. Kent, B. A. Olshausen, E. Osipov, J. M. Rabaey, D. A. Rachkovskij, A. Rahimi, F. T. Sommer, *arXiv:2106.05268* **2021**.
- [33] C. Xu, S. Chen, J. Su, S. M. Yiu, L. C. K. Hui, *IEEE Commun. Surv. Tutorials* **2016**, 18, 2991.
- [34] M.-F. Chang, C.-C. Lin, A. Lee, Y.-N. Chiang, C.-C. Kuo, G.-H. Yang, H.-J. Tsai, T.-F. Chen, S.-S. Sheu, *IEEE J. Solid-State Circuits* **2017**, 52, 1664.
- [35] A. Grossi, E. Vianello, C. Zambelli, P. Royer, J.-P. Noel, B. Giraud, L. Perniola, P. Olivo, E. Nowak, *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **2018**, 26, 2599.
- [36] C. E. Graves, S.-T. Lam, X. Li, L. Kiyama, M. Foltin, M. P. Hardy, J. P. Strachan, C. Li, X. Sheng, W. Ma, S. R. Chalamalasetti, D. Miller, J. S. Ignowski, B. Buchanan, L. Zheng, *IEEE Trans. Nanotechnol.* **2019**, 18, 963.
- [37] C. E. Graves, C. Li, X. Sheng, D. Miller, J. Ignowski, L. Kiyama, J. P. Strachan, *Adv. Mater.* **2020**, 32, 2003437.
- [38] H. Li, W.-C. Chen, A. Levy, C.-H. Wang, H. Wang, P.-H. Chen, W. Wan, W.-S. Khwa, H. Chuang, Y.-D. Chih, M.-F. Chang, H.-S. P. Wong, P. Raina, *IEEE Trans. Electron Devices* **2021**, 68, 6637.
- [39] X. Wang, L. Wang, Y. Wang, J. An, C. Dou, Z. Wu, X. Zhang, J. Liu, C. Zhang, Z. Yao, Z. Yu, T. Shi, C. Chen, X. Jiang, M.-F. Chang, Q. Liu, *IEEE Trans. Electron Devices* **2021**, 68, 5.
- [40] K. Ni, X. Yin, A. F. Laguna, S. Joshi, S. Dünkel, M. Trentzsch, J. Müller, S. Beyer, M. Niemier, X. S. Hu, S. Datta, *Nat. Electron.* **2019**, 2, 521.
- [41] C. Li, F. Muller, T. Ali, R. Olivo, M. Imani, S. Deng, C. Zhuo, T. Kampfe, X. Yin, K. Ni, in *2020 IEEE Int. Electron Devices Meeting (IEDM)*, IEEE, Piscataway, NJ **2020**, pp. 29.3.1–29.3.4.
- [42] C. Li, C. E. Graves, X. Sheng, D. Miller, M. Foltin, G. Pedretti, J. P. Strachan, *Nat. Commun.* **2020**, 11, 1638.
- [43] G. Pedretti, C. E. Graves, S. Serebryakov, R. Mao, X. Sheng, M. Foltin, C. Li, J. P. Strachan, *Nat. Commun.* **2021**, 12, 5806.
- [44] X. Sheng, C. E. Graves, S. Kumar, X. Li, B. Buchanan, L. Zheng, S. Lam, C. Li, J. P. Strachan, *Adv. Electron. Mater.* **2019**, 5, 1800876.
- [45] C. Li, J. Ignowski, X. Sheng, R. Wessel, B. Jaffe, J. Ingemi, C. Graves, J. P. Strachan, in *2020 IEEE Int. Memory Workshop (IMW)*, IEEE, Piscataway, NJ **2020**, pp. 1–4.
- [46] G. Pedretti, D. Ielmini, *Electronics* **2021**, 10, 9.
- [47] A. Bandyopadhyay, G. J. Serrano, P. Hasler, in *2005 IEEE Int. Symp. on Circuits and Systems*, IEEE, Piscataway, NJ **2005**, pp. 2148–2151.
- [48] F. Alibart, L. Gao, B. D. Hoskins, D. B. Strukov **2012**, 23, 7 075201
- [49] I. Boybat, M. Le Gallo, S. R. Nandakumar, T. Moraitis, T. Parnell, T. Tuma, B. Rajendran, Y. Leblebici, A. Sebastian, E. Eleftheriou, *Nat. Commun.* **2018**, 9, 2514.
- [50] G. Pedretti, E. Ambrosi, D. Ielmini, in *2021 IEEE Int. Reliability Physics Symp. (IRPS)*, IEEE, Piscataway, NJ **2021**, pp. 1–8.
- [51] G. Pedretti, P. Mannocci, C. Li, Z. Sun, J. P. Strachan, D. Ielmini, *IEEE Trans. Electron Devices* **2021**, 68, 4373.
- [52] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, S. Chintala, in *Advances in Neural Information Processing Systems*, (Eds: H. Wallach, H. Larochelle, A. Beygelzimer, F. d. Alché-Buc, E. Fox, R. Garnett) Vol. 32, Curran Associates, Red Hook, NY **2019**.
- [53] R. A. Fisher, *Ann. Eugenics* **1936**, 7, 179.
- [54] Y. Ye, S. Gummalla, C.-C. Wang, C. Chakrabarti, Y. Cao, *J. Comput. Electron.* **2010**, 9, 108.
- [55] R. Impagliazzo, R. Paturi, *J. Comput. Syst. Sci.* **2001**, 62, 367.
- [56] L. Fortnow, *Commun. ACM* **2009**, 52, 78.
- [57] J. P. Strachan, C. E. Graves, K-SAT filter querying using ternary content-addressable memory, **2020**.
- [58] H. H. Hoos, T. Stützle, in *Frontiers in Artificial Intelligence and Applications*, IOS Press, Amsterdam **2000**, pp. 283–292.
- [59] B. Selman, D. G. Mitchell, H. J. Levesque, *Artif. Intell.* **1996**, 81, 17.
- [60] A. Lucas, *Front. Phys.* **2014**, 2, 5.
- [61] T.-H. Chen, J.-Y. Lu, *arXiv:1603.05314* **2016**.
- [62] G. Pedretti, P. Mannocci, C. Li, Z. Sun, J. P. Strachan, D. Ielmini, *IEEE Trans. Electron Devices* **2021**, pp. 1–6.