A Model-to-Circuit Compiler for Evaluation of DNN Accelerators based on Systolic Arrays and Multibit Emerging Memories

Johannes Knödtel*, Markus Fritscher*, Daniel Reiser*, Dietmar Fey*, Marco Breiling†, Marc Reichenbach*
*Chair for Computer Architecture, Friedrich-Alexander-Universität Erlagen-Nürnberg
†Fraunhofer Institute for Integrated Circuits
*{johannes.knoedtel,markus.fritscher,daniel.g.reiser,dietmar.fey,marc.reichenbach}@fau.de
†marco.breiling@iis.fraunhofer.de

***Abstract*—With the emergence of DNN accelerators the main focus of such systems usually lies on utilizing local memories and reducing the size of the processed data, since delay and energy consumption are dominated by data transfer. Utilizig emerging memory technologies, such as ReRAMs, these goals might be attained much easier, due to advantageous non-functional and functional properties.**

One of the key drawbacks of such systems are reliability and variability of devices of such technologies. To certain degree, DNNs are resilient to soft and hard errors in their memory cells, so these issues might be surmountable depending on the device properties, but eludes trivial analyses known from the digital domain. Here the dynamic behavior of the devices comes into play and must be simulated in order to get a decent degree of confidence on the reliability of the hardware and therefore also yield.

In order to tackle this issue we present an accelerator architecture and matching analysis pipeline that allows the user to specify and train a net topology and then test the design against some input activations with different randomized device properties. Using this approach we can estimate the inference results and other system and algorithm level properties in presence of different device level properties which might for example be extracted from real world measurements. Such a system can help the user in the design of their net or give them hints on the required device properties for a given net or aid them in evaluating existing designs.

I. Introduction

Artificial intelligence by the usage of deep neuronal networks is one of the most important research area in the recent years[3]. Apart from algorithmic improvements, many different new hardware architectures were developed to speed-up the process of inference. Examples include GPUs (e.g. [11, 13]) as well as dedicated neuromorphic hardware accelerators such as Google's TPU[6].

For developing such new accelerators one of the main concepts is to maximize the energy efficiency, meaning to perform high speed inference, with billions of MAC operations (multiply and accumulate) while maintaining a minimum power budget. While the calculations itself could be implemented with dedicated hardware architectures, one of the crucial tasks is minimizing external memory access and maintaining data stationarity. This has been the prime target of quite a few accelerator architectures such as the Eyriss architecture[1] or the COSY architecture[19].

When performing a neuronal network inference using a matrix vector multiplications, with a weight matrix W and input vector x, a major part of the energy will be spent while transferring the weights $w_{i,j}$ from external memory in the chip. Therefore, one idea is, to store as much as possible weights inside the chip, to avoid external memory access. Emerging memories might prove beneficial here given their differing properties, e.g. non-volatility in memristive devices.

Moreover, since many technologies offer the possibility to store more than 2 states in one memory cell (e.g. ReRAM), this kind of memory is a good candidate for highly quantized neuronal networks, which makes it possible to store exactly one weight $w_{i,j}$ in one ReRAM cell. For example research has shown that ternary network can be very effective in solving different tasks, as described in [8].

Using new memory technologies can lead to extensive redesigns of the hardware architecture, if technology or technology parameters are subject to changes. For example, if cycle-to-cycle variability is reduced, the possibility of storing less quantized parameters might become viable. To capitalize on this advantage, readout circuits need to adapted and the net could potentially be reduced to a smaller one. Algorithmic and system level properties are therefore heavily influenced by device level properties.

Therefore, we propose in this paper a new design methodology by providing model-to-circuit compiler, which automatically derives a circuit from a given neuronal network model. Due to the possibility of a mixed-signal evaluation, we are able to evaluate functional and non-functional properties.

Many different use cases for such a flow are immediately apparent:

- Given a set of real world devices with different characteristics, one can investigate which are a viable candidates for a given net.
- Finding which nets typologies are suitable for a combination of a given task and device.
- Investigating which variation parameters are still acceptable or will affect yield.

While it is possible to investigate this using traditional flows, doing so for e.g. the feedback loops, where one tries to tie system level properties to other parameters, are tedious and include a lot of designer input. With our approach the generation of designs and evaluation of these designs under some defined device level properties is done mostly in an automated fashion.

Summarizing, the contributions of this paper are:

- A generic architecture, based on systolic arrays, as a general IP-block for building hardware accelerators for fast and energy efficient neuronal network inference.
- A model-to-circuit compiler which uses this architecture as a template and adapts it to the given neuronal network model to instantiate said architecture.
- A mixed-signal simulation environment, which allows to automatically evaluate the created architectures. This will allow for an automated design-space exploration as a long time goal.

The paper is structured as follows. In this Section we presented the importance and the contribution of our work. Next Section will discuss related work and basic principles for our architecture. In Section III we will present the methodology, meaning the generic architecture, the model-to-circuit-compiler and the mixed-signal simulation environment. Next, we show our methodology by example of a simple multi-layer net, showing results and discussing it. Finally, in Section V we will conclude our paper and give a outlook of future work.

II. Fundamentals

In this section we will discuss the fundamentals of the technologies that are used to test and evaluate our methodology. In particular this will be systolic arrays, ReRAMs and design considerations when combining these technologies.

A. ReRAM

Many researchers highlight the ability to push the boundaries set by traditional memory technologies with memristive memories. One instance of memristive devices are ReRAMs. These devices are in essence memories that store data using their resistive state. ReRAMs, as can be seen in Fig. 1a, are usually manufactured as metal-insulator-metal structures that are, similar to MIM capacitors, CMOS compatible and can be manufactured in the processing line.

The operation of ReRAM devices is as follows:

a) Read: Using a small voltage, the current resistance value is determined, representing the state of the devices.

b) Write: Using a larger negative or positive voltage, the device state is altered. In ECM-based technologies, a common type of ReRAMs, this amounts to the formation of a filament structure with lower resistance as the surrounding insulator (Fig. 1b). The length of the filament and therefore the resistance can be coarsely modulated, leading to the ability to store more than one bit of information. Researchers have shown, that up to 6.5 bits of information per cell are possible[17]. For achieving these states in practice, different techniques exist, such as current compliance or pulsing [15].

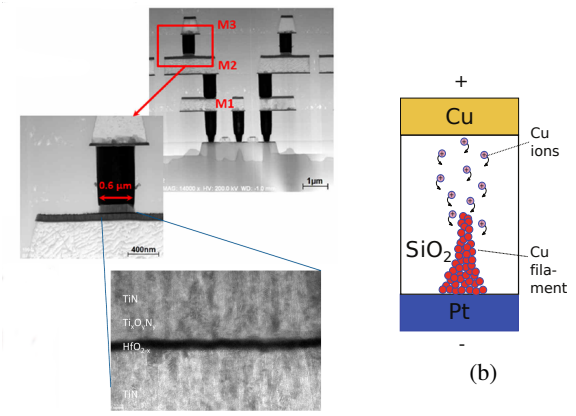


Fig. 1: Subfigure (a) shows a MIM stack of a HfO2-based cell with TiN electrodes. It is placed on top of a regular MOS transistor which acts as a selector device. Such a setup is referred to as a 1T1R device. Subfigure (b) shows an idealized representation of an ECM-based ReRAM device. Given a high enough voltage across its terminals, it as able to grow or retract a filament composed of Cu-Ions from the top electrode. The state of the device is mainly determined by the geometry of the filament - the distance between top electrode and filament is considered as the modulated variable for achieving different resistance levels.

The most critical aspect in this is to deal with the variability of the devices[10].

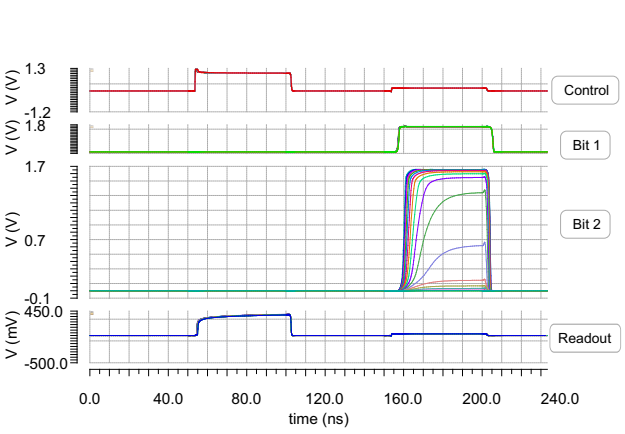
B. Systolic Arrays

In order to leverage these technologies we decided to investigate these technologies in weight storage for a systolic array which implements matrix-vector multipliers for large input sizes. Systolic arrays are regular structures for computing tasks on matrices presented by Kung and Leiserson in 1978[7] and are a standard component of most DNN accelerators due to their inherent parallelism and advantages for energy and latency efficient layouting in ASICs. The general idea is illustrated in Fig. 3: The input vector is fed in from the left where it is multiplied with a weight and also passed unaltered onto the next column. A cell performing such a computation is referred to as a Processing Unit (PU). The result is passed down to the next row, where it is added to its multiplication; this is called a multiply-accumulate (MAC). Using this technique an array of $n \times m$ can calculate the result in $n + m$ cycles. For larger inputs partial results need to be accumulated (as depicted with ACC units).

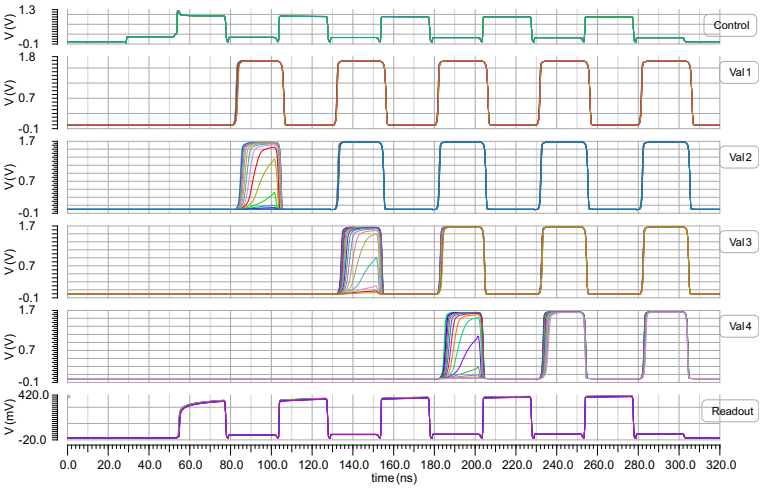
The usage of novel memory technologies was already explored in [9] with a focus on energy and area savings. In this work we are focusing on functional design criterions:

C. Design Considerations

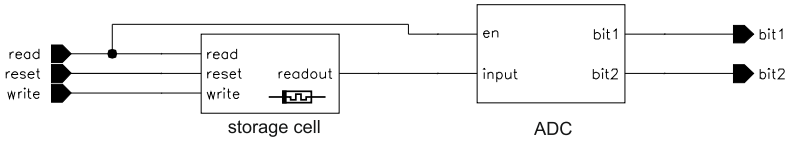
Since the devices exhibit device-to-device (D2D) and cycle-to-cycle variability, depending mainly on technology and manufacturing, one needs to employ measures to limit the impact of the variations. This variability has been target of extensive studies, such as [14], due to its impact on performance when



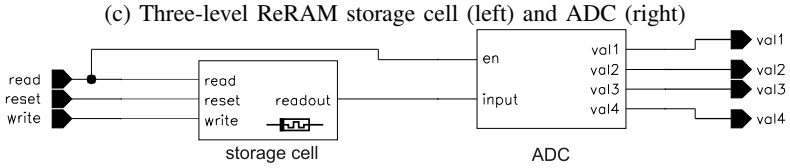
(a) Parametric simulation: three-level storage cell



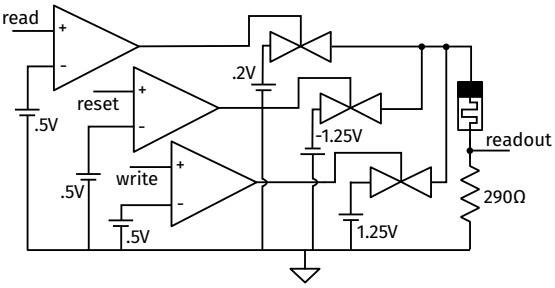
(b) Parametric simulation: five-level storage cell



(c) Three-level ReRAM storage cell (left) and ADC (right)



(d) Five-level ReRAM storage cell (left) and ADC (right)



(e) Storage cell as shown in Fig. 2c

Fig. 2: This figure shows the analog storage cells embedded in our Mixed-Signal-Simulation. The top-level view of the three-level cell is depicted in Fig. 2c while the top-level view of the five-level cell is shown in Fig. 2d. It is controlled by a memory controller, which provides the digital control signals $read$, $reset$ and $write$ and which reads the digitized values from $bit1/bit2$ respectively $val1-4$. The five-state ADC utilizes thermometer-encoding as it is easy to work with. The implementation of the storage cell itself can be seen in Fig. 2e. It applies a read ($200\,\mathrm{mV}$), reset ($-1.25\,\mathrm{V}$) or write ($1.25\,\mathrm{V}$) pulse to the top electrode of a ReRAM device when a control signal is generated. The signal at the bottom electrode doubles as the input for the ADC circuit (and is depicted in $readout$). Fig. 2a and Fig. 2b show waveforms generated by running a parametric simulation performing cycles of reading and writing increasing values into the cell. Pulses with small amplitudes in the control and readout signals correspond to read pulses, to generate a voltage for the ADC to read the current resistance value and larger amplitudes to write pulses, which should alter the resistance state of the device. Successive pulses should in an ideal scenario alter the device in a way that lead in the following reads to a increased value. This value is determined during read pulses from small differences in voltage at the readout node. We varied the oxide thickness in different linearly spaced steps, ranging from $6.45\,\mathrm{nm}$ to $6.6\,\mathrm{nm}$ to determine whether a device's variance impacts the performance. One can see drastic differences in the resulting values for some of the simulations, the device is not reliably set, an erroneus value is read back. This is reflected by the bit/val values, not reaching the matching voltages during the read phase. The simulation uses the Stanford ReRAM model[5].

used in digital systems. An example for these variations can be seen in Fig. 2. This work tries to enable the user to determine whether a neural network embedding such storage cells suffering from variances is still able to perform in the same way. In the worst case scenario, the write circuitry is unable to correctly alter device state or the read circuitry is unable to read the correct value. This forces the designer to consider error correction, error tolerant logic/algorithms or some circuit-level mitigation (e.g. [12]).

Since DNNs are known to exhibit some inherent redundancy, calculation errors might be tolerable. In [16] a more detailed analysis of this phenomenon can be found and in [2] it is explored how sensitive to errors individual weights are.

The aspect of error tolerance is one of the key components of our approach. Using device models with randomly altered parameters we can simulate such variabilities and determine if a given net will still yield correct results. Our automated flow handles everything from training the net down to a mixed-signal simulation based on an inferred netlist and a testbench with weight and given activations.

III. Methodology and Implementation

Our approach to investigate this issue is a pipeline from a neural net specification to a mixed signal simulation. The block diagram in Fig. 4 illustrates the flow we will describe in the following subsections. In principle it can be divided into three stages:

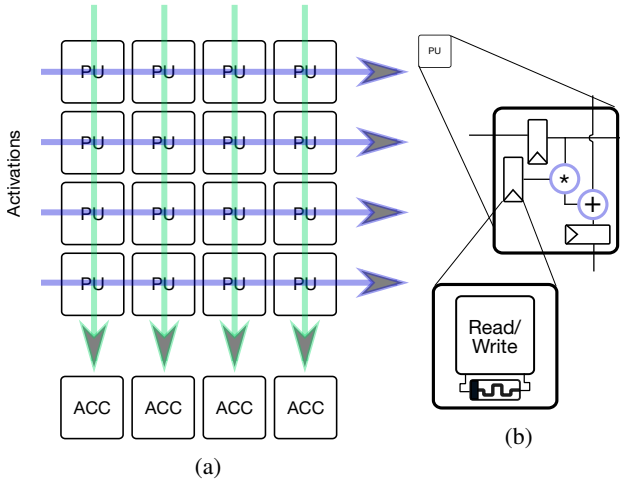


Fig. 3: Subfigure (a) shows the construction of a systolic array. Activations, which are either the input to the algorithm, or passed down from a layer higher up, are passed from left to right in a pipelined fashion. In each Processing Unit (PU) the activation is multiplied and added to the partial sum that is passed down from the cell one row higher up. The accumulators (ACC) at the bottom edge sum up the incoming partial sums from above. Subfigure (b) details the composition of a PU cell. It consists of three building blocks: (1.) pass-through register, which builds up the Pipeline the forwards the activations across columns. (2.) weight register, which stores the multiplicand in the MAC operation, which in our case is a multilevel ReRAM cell and auxiliary circuitry for read and write (3.) arithmetic components, which are a multiplier and adder which implement the MAC operation.

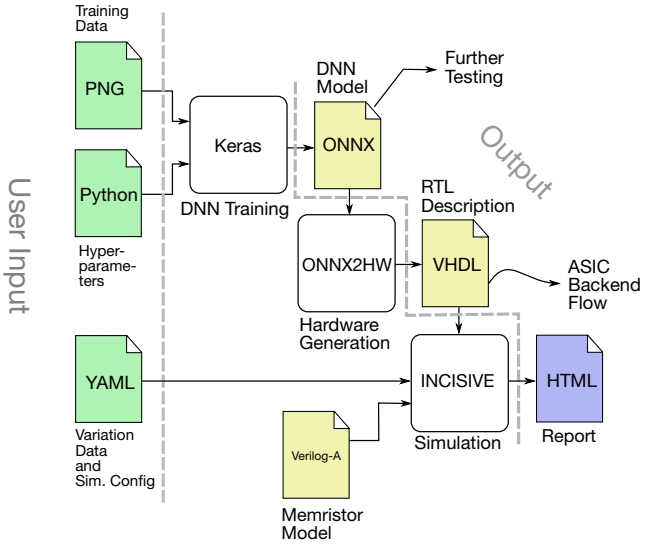


Fig. 4: This block diagram illustrates the implemented flow. User defined inputs are on the left, while output relevant to the designer is on the right. The end goal of the simulation runs is a report on the reliability in presence, which is shown in detail in Fig 5.

1) Net Training
2) Hardware Generation
3) Simulation/Evaluation

In this section we will cover these stages.

A. Training/Model-to-Netlist Transformation

The concept of hardware generators and generic configurable designs for DNN accelerators was already investigated by other researchers (e.g. [18], or as open source software [20]), but our flow relies on our own hardware generator, that is adapted to this specific usecase which is not supported by traditional generators. It abstracts the weight memory in such a way that it can be implemented using different technologies, supporting arbitrary quantizations, not necessarily relying on regular binary memories. The focus of the generator is not to generate the best real world systems, but the exploration of the design space; For readers interested in current developments in this area, [21] provides a good overview.

The topological features of the neural net are given by the user in form of Keras program. This representation was chosen since most models are derived from some program using standard toolkits, such as PyTorch, Keras or Tensorflow. For the quantization of the weights specialized code is provided which has little overhead when integrating it into existing Keras programs. This custom code is specifically designed to adapt to the needs in multi-level memories, since we are not only covering the case of reduced bits, but for arbitrarily quantized values (e.g. a user can specify, that exactly five values shall be used, which is not power of two range of a normal binary number). The main advantage of this approach lies within its flexibility: Many existing programs can be used with slight modifications; Although it might have been more general to rely on given exchange formats at this stage and let the user output their nets to these, we opted to integrate training in our flow, since quantization can be flexibly introduced at any stage of the generation. The user has the option to either quantize after training, or train with quantized weights.

After the training the model is exported into ONNX, a graph-based file format for neural nets and then passed onto our custom hardware generator. This generator converts the given nodes into hardware components and wires them accordingly using VHDL as the HDL. The generator does not generate any code for the weight storage, this component is provided by the user. This lets the user pick any readout circuitry and device models for the simulation, as long as the appropriate interface is kept. These components are required to model the variations by reading parameters from text files. Since most models are written in Verilog-A, this is fairly easy, because the language and most simulators support text I/O.

B. Simulation Setup

For modeling the variation for a given distribution each analog memory device is provided with a random initialization of user chosen parameters. Subsequently, multiple runs with different values are executed in order to evaluate how sensitive the generated hardware embedding these memory devices is to these variations. This allows for the evaluation whether the hardware is able to cope with the resulting defects and variations. We used the mixed-signal environment described in

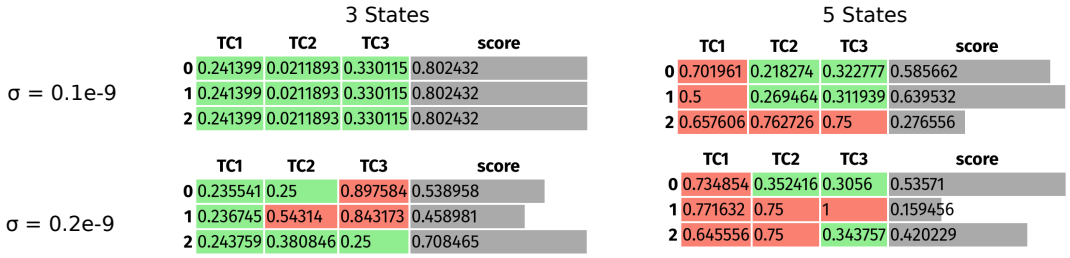


Fig. 5: This is the way results are presented by our automated flow. The tables show a comparison of a net with 3 possible states in the weight memory to a net with 5 possible states, along with two different distributions of variation parameters. Each table corresponds to a variation distribution for a given net, with each row corresponding to a sample drawn from this distribution. The columns labeled "TC" represent a different activation as test input to the network. Both networks share the same topology, training data and number of training epochs. The device model used was (as with Fig. 2) the Stanford ReRAM model with normally distributed variations in the T_{ox} variable. The mean of the distribution was the default value, while the standard deviation is given in the figure. For this classification task based on a number pattern, we opted, in order to highlight the differences in presence of variations, to use the angle between expected and actual output vector normalized to the range of $[0; 1]$ (< 0.5 indicates correct outputs, incorrect otherwise). This way one can notice finer details, when the net still classifies the input correctly according to the class with the highest output value. A score is calulated to quickly estimate the reliability of the resulting hardware by averaging the aforementioned values. For brevity only three sets of variations and three test case activations each are included.

```
  [...]
sim:
  [...]
  activations:
    - file: act0.csv
      expect: [1,0,0,0,0,0,0,0,0,0]
    - file: act1.csv
      expect: [0,1,0,0,0,0,0,0,0,0]
  [...]
  variations_samples: 10
  variations:
    - parameter: tox
      type: gaussian
      mean: 1.19
      stddev: 0.1
  [...]
```

Fig. 6: An example for simulation configuration is shown. For the simulation different activations can be specified as Pass/Fail-type tests. In this example a net for the MNIST dataset is investigated. The activations section specifies different activations by stating a file and an expected output of the net. If this output does not match the expected value, the test is considered a failure. Parameters subject to variations can be specified in the variations section. These are randomly drawn from a user specified distribution such as a Gaussian or a log-normal distribution.

[4] to run these simulations. How these effects can be observed is detailed in our discussion of the results in Section IV.

C. Integration

The flow is integrated in order to be useful for large-scale design space explorations. A configuration is provided by the user specifying the device level properties as well as some technical parameters relating to the execution of used tools. An example for this can be seen in Fig. 6. In a design space exploration different variations can be tested. This data can then be used to investigate different aspects, as described previously.

IV. Discussion

We found that this approach yields a tool that is suitable for the task. The results show different failure characteristics for different variation settings. Our tool generates a table of results for different runs and test cases. It can for example be investigated, whether certain classes of input data will consistently fail with a certain type and magnitude of variation.

In Fig. 5 a report for an example net can be seen. In this example we trained two networks on a simple binary classification problem. One utilizes ternary while the other uses quinary weights. The network utilizing quinary weights yields an accuracy of 98.7% while the other network settles at 72.7% as evaluated by Keras after training. However, as shown before, a storage cell with less quantized weights might be more susceptible to variations and, ultimately, provide worse accuracy. Ultimately, we could find that the three weight state net was less susceptible to variations in general and outperformed the quinary net. Additionally we could determine other effects: The color coding of the tables lets the user quickly see which activations for are affected which net in magnitude. Red fields indicate a failed test case for the net with a certain set of variation parameters, while green fields indicate a passing one. We can see that the quinary net has issues to correctly classify test case no. 1; we also observed that for the ternary net, a low variation yielded completely identical and correct results for all testcases under all variations.

We plan on larger investigations on well known networks and network topologies as future work.

V. Conclusion

In this work we have shown the use cases and benefits of determining the system and algorithm level effects of device properties in emerging multibit memories using an automated custom flow created specifically for this task. Our approach lends itself well for design space exploration in this field, as well as some potential future work. For example, this approach could automatically assist in "retraining" the net when the effects of variations are apparent, or for estimating the effects of switching to a different memory technology.

References

[1] Yu-Hsin Chen et al. "Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks". In: *IEEE Journal of Solid-State Circuits* 52.1 (Jan. 2017), pp. 127–138. ISSN: 1558-173X. DOI: 10.1109/JSSC.2016.2616357.

[2] Wonseok Choi et al. "Sensitivity Based Error Resilient Techniques for Energy Efficient Deep Neural Network Accelerators". In: *Proceedings of the 56th Annual Design Automation Conference 2019*. DAC '19. Las Vegas, NV, USA: Association for Computing Machinery, 2019. ISBN: 9781450367257. DOI: 10.1145/3316781.3317908. URL: https://doi.org/10.1145/3316781.3317908.

[3] Katie Costello. *Gartner Predicts the Future of AI Technologies*. URL: https://www.gartner.com/smarterwithgartner/gartner-predicts-the-future-of-ai-technologies/ (visited on 01/22/2020).

[4] Markus Fritscher et al. "Simulating Memristive Systems in Mixed-Signal Mode using Commercial Design Tools". In: International Conference on Electronics Circuits and Systems (ICECS). (to be published). Genova, Italy, Nov. 27, 2019.

[5] Ximeng Guan, Shimeng Yu, and H. S Philip Wong. "A SPICE compact model of metal oxide resistive switching memory with variations". English (US). In: *IEEE Electron Device Letters* 33.10 (2012), pp. 1405–1407. ISSN: 0741-3106. DOI: 10.1109/LED.2012.2210856.

[6] Norman P. Jouppi et al. "In-datacenter performance analysis of a tensor processing unit". In: *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)* (2017), pp. 1–12. DOI: 10.1145/3079856.3080246.

[7] HT Kung and Charles E Leiserson. "Systolic arrays (for VLSI)". In: *Sparse Matrix Proceedings 1978*. Vol. 1. Society for industrial and applied mathematics. 1979, pp. 256–282.

[8] Fengfu Li and Bin Liu. "Ternary Weight Networks". In: *ArXiv* abs/1605.04711 (2016).

[9] H. Li et al. "On-Chip Memory Technology Design Space Explorations for Mobile Deep Neural Network Accelerators". In: *2019 56th ACM/IEEE Design Automation Conference (DAC)*. 2019.

[10] Tobias Lieske, Mehrdad Biglari, and Dietmar Fey. "Multi-Level Memristive Voltage Divider: Programming Scheme Trade-Offs". In: *Proceedings of the International Symposium on Memory Systems*. MEMSYS '18. Alexandria, Virginia, USA: Association for Computing Machinery, 2018, pp. 259–268. ISBN: 9781450364751. DOI: 10.1145/3240302.3240430. URL: https://doi.org/10.1145/3240302.3240430.

[11] Sparsh Mittal and Shraiysh Vaishay. "A Survey of Techniques for Optimizing Deep Learning on GPUs". In: *Journal of Systems Architecture* 99 (), p. 101635. ISSN: 1383-7621. URL: https://www.academia.edu/40135801/A_Survey_of_Techniques_for_Optimizing_Deep_Learning_on_GPUs (visited on 01/23/2020).

[12] Dimin Niu, Yang Xiao, and Yuan Xie. "Low power memristor-based ReRAM design with Error Correcting Code". In: *17th Asia and South Pacific Design Automation Conference*. 17th Asia and South Pacific Design Automation Conference. Jan. 2012, pp. 79–84. DOI: 10.1109/ASPDAC.2012.6165062.

[13] Kyoung-Su Oh and Keechul Jung. "GPU implementation of neural networks". In: *Pattern Recognition* 37.6 (June 1, 2004), pp. 1311–1314. ISSN: 0031-3203. DOI: 10.1016/j.patcog.2004.01.013.

[14] E. Pérez et al. "Analysis of the statistics of device-to-device and cycle-to-cycle variability in TiN/Ti/Al:HfO2/TiN RRAMs". In: *Microelectronic Engineering* 214 (June 2019), pp. 104–109. ISSN: 01679317. DOI: 10.1016/j.mee.2019.05.004. (Visited on 01/24/2020).

[15] Eduardo Pérez et al. "Towards Reliable Multi-Level Operation in RRAM Arrays: Improving Post-Algorithm Stability and Assessing Endurance/Data Retention". In: *IEEE Journal of the Electron Devices Society* PP (July 2019), pp. 1–1. DOI: 10.1109/JEDS.2019.2931769.

[16] Brandon Reagen et al. "Ares: A framework for quantifying the resilience of deep neural networks". In: *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*. 2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC). June 2018, pp. 1–6. DOI: 10.1109/DAC.2018.8465834.

[17] Spyros Stathopoulos et al. "Multibit memory operation of metal-oxide bi-layer memristors". In: *Scientific Reports* 7.1 (Dec. 13, 2017), pp. 1–7. ISSN: 2045-2322. DOI: 10.1038/s41598-017-17785-1. URL: https://www.nature.com/articles/s41598-017-17785-1 (visited on 01/24/2020).

[18] Xuechao Wei et al. "Automated Systolic Array Architecture Synthesis for High Throughput CNN Inference on FPGAs". In: June 18, 2017, pp. 1–6. DOI: 10.1145/3061639.3062207.

[19] Chen Xin et al. "COSY: An Energy-Efficient Hardware Architecture for Deep Convolutional Neural Networks Based on Systolic Array". In: *2017 IEEE 23rd International Conference on Parallel and Distributed Systems (ICPADS)*. 2017 IEEE 23rd International Conference on Parallel and Distributed Systems (ICPADS). Dec. 2017, pp. 180–189. DOI: 10.1109/ICPADS.2017.00034.

[20] Jerry Zhao. *jerryz123/onnx-halide*. original-date: 2018-11-12T21:05:45Z. Aug. 15, 2019. URL: https://github.com/jerryz123/onnx-halide (visited on 01/22/2020).

[21] R. Zhao et al. "Hardware Compilation of Deep Neural Networks: An Overview". In: *2018 IEEE 29th International Conference on Application-specific Systems Architectures and Processors (ASAP)*. 2018.