RRAM-VAC: A Variability-Aware Controller for RRAM-based Memory Architectures

Shikhar Tuli, Marco Rios, Alexandre Levisse and David Atienza
Embedded System Laboratory (ESL), Ecole Polytechnique Fédérale de Lausanne (EPFL), Switzerland.
alexandre.levisse@epfl.ch

ABSTRACT

The growing need for connected, smart and energy efficient devices requires them to provide both ultra-low standby power and relatively high computing capabilities when awoken. In this context, emerging resistive memory technologies (RRAM) appear as a promising solution as they enable cheap fine grain technology co-integration with CMOS, fast switching and non-volatile storage. However, RRAM technologies suffer from fundamental flaws such as a strong device-to-device and cycle-to-cycle variability which is worsened by aging, forcing the designers to consider worst case design conditions. In this work, we propose, for the first time, a circuit that can take advantage of recently published Write Termination (WT) circuits from both the energy and performances point of view. The proposed RRAM Variability Aware Controller (RRAM-VAC) stores and then coalesces the write requests from the processor before triggering the actual write process. By doing so, it averages the RRAM variability and enables the system to run at the memory programming time distribution mean rather than the worst case tail. We explore the design space of the proposed solution for various RRAM variability specifications, benchmark the effect of the proposed memory controller with real application memory traces and show (for the considered RRAM technology specifications) 44 % to 50 % performances improvement and from 10% to 85% energy gains depending on the application memory access patterns.

KEYWORDS

Non-Volatile Memories, RRAM, Memory Controller, Resistive Memories, Edge Computing

ACM Reference Format:
Shikhar Tuli, Marco Rios, Alexandre Levisse and David Atienza. 2019. RRAM-VAC: A Variability-Aware Controller for RRAM-based Memory Architectures. In *xxxx*. ACM, New York, NY, USA, 6 pages. https://doi.org/xxxx

1 INTRODUCTION

The ever-increasing number of Internet of Things devices present in our lives is forcing a shift in the computational paradigm. Instead of centralizing the processing in big data centers, modern applications are seeking to compute the data locally on the edge in order to improve latency, energy efficiency, privacy and security. From that perspective, the market of connected health monitoring is a good candidate as it features a large amount of critical data which must be computed precisely, timely and efficiently. In that context, Wireless Body Sensor Node (WBSNs) are expected to switch from a zero-leakage idle deep-sleep mode (which can happen more than 90% of the device lifetime) to a relatively high performances computation phase during which, in order to mitigate the leakage power, all the computation must be performed timely. From that perspective, emerging Resistive Random Access Memory (RRAM) technologies appear as a good fit, thanks to, compared to eflash technologies, their cheap and easy technology co-integration within CMOS process, their fast switching capabilities and low voltage operation [3, 6, 15, 16]. However, their drawbacks can only be slightly mitigated by device engineering and must be solved by design. For instance, device-to-device and cycle-to-cycle temporal variability may lead to extremely dispersed programming times (several decades [11]) and can only be managed by the use of Write Termination (WT) circuits [2, 8, 11]. Although these works focus on energy reduction, they do not propose any solution for the performances improvements that could be enabled by such circuits (i.e., these solution force the system to run at the worst case memory frequency) and mainly focus on circuit considerations. In this work, we propose the concept of RRAM Variability-Aware Controller (RRAM-VAC). The RRAM-VAC stores the *write* requests (i.e. *data* and *addresses*) from the processor and coalesces them before writing them to memory. By doing so, it averages the programming time and can theoretically improve the performances from the worst case programming time down to the average programming time of the distribution. In this context, it could enable substantial static energy gains. We validate the RRAM-VAC functionality by implementing a behavioral model and explore its functionality with realistic technology and circuit assumptions. Then we benchmark system-level gains by simulating it with memory traces from WBSN applications and comparing it to a reference case without WT and RRAM-VAC. The contributions of the paper are as follows:

- We propose the concept of RRAM-VAC controller and validate its functionality through a behavioral model supported by accurate RRAM technology and circuit assumptions.
- We explore the design space of the RRAM-VAC and propose a sizing methodology for various parameters such as the operating frequency and the RRAM variability parameters.
- We simulate the RRAM-VAC controller with realistic memory traces from WBSN applications and show that for the considered technology assumptions it enables up to 50% performance improvement and from 10 to 85% energy reduction depending on the application memory access patterns.

The rest of the paper is organized as follows. Section 2 includes a general background on RRAM technology, architecture and writing

ASP-DAC'20, Jan. 13-16, 2020, Beijing, China

ACM ISBN xxxx.
https://doi.org/xxxx

circuitry. Section 3 presents the proposed RRAM-VAC architecture. Section 4 presents the experimental setup used for the simulations. Section 5 presents a design space exploration and shows performance and energy gains enabled by the RRAM-VAC. Finally, Section 6 concludes the paper.

2 BACKGROUND

2.1 RRAM Technologies for Embedded Devices

With the widespread of Internet of Things connected Edge devices, requirements in terms of price per device and energy efficiency have been rising. In this context, new Resistive Memory (RRAM) technologies have been proposed to replace regular eflash technologies and are already at the product maturity [7, 10, 14]. RRAM technologies rely on the non-volatile variation of the resistivity of a thin insulating layer between Low and High Resistance States (LRS, HRS). This effect is achieved through various mechanisms such as (i) a conductive ions migration inside an insulating layer (Resistive RRAM - ReRAM) [6, 16], (ii) a phase change inside a chalcogenide material (Phase Change Memories - PCM) [15] or (iii) a spin modification in a magnetic tunnel junction (Magnetic Memories - MRAM) [3]. Their cheap and easy technology co-integration with CMOS, low programming voltages (1 to 3V) and fast switching capabilities (tens to hundred of ns)) triggered the motivation to abandon eflash beyond the 28nm node technology. Embedded RRAM memory arrays are usually constituted of 3-terminals 1 Transistor - 1 RRAM bitcells, controlled by a WordLine (WL), and SourceLine (SL) and and BitLine (BL). *Read* and *write* operations are controlled from the BL and SL thanks to specialized circuits named Sense Amplifier (SA) for *read* and Write Amplifier (WrA) for *write*. *Read* and *write* operation in RRAM-based array are highly asymmetric : one way of performing *read* is by pre-charging the BL and discharging it through the RRAM bitcell. Then the resulting BL voltage is amplified and read-out by the SA. On the other hand, *write* operations are performed by applying a high enough voltage programming pulse and limiting the current. In that context, *read* operations are mainly dynamic power plus the SA overhead while *write* operations feature high static consumption leading to one operation being more energy hungry than the other.

2.2 Write termination circuits

Due to the complex underlying physics, RRAM technologies, as a whole, suffers from a high device-to-device and cycle-to-cycle variability [3, 16]. This effect is particularly true in filamentary RRAM technologies due to the stochastic nature of ions movement inside the insulator [11]. In this context, usual write methods, simply consist in applying a long-enough programming pulse to cover the complete distribution of programming time [11]. To overcome this issue, Write Termination (WT) WrA circuits have been proposed [2, 8, 11], they consist of a dynamic detection of the current flowing through the RRAM and feature a detection mechanism stopping the write operation when the current crosses a given threshold. These solution come in addition to already widely reported write-verify solutions that consists in checking the state of the RRAM after a complete programming pulse and restarting if needed [7, 14]. However, for all these solutions, while the programming energy can be cut drastically, there are no reported solutions taking advantage of the temporal variability to enhance the performances of embedded systems. In this work, we thereby propose a specific memory controller which enables both energy and performance improvements for RRAM-based embedded devices: the RRAM-VAC.

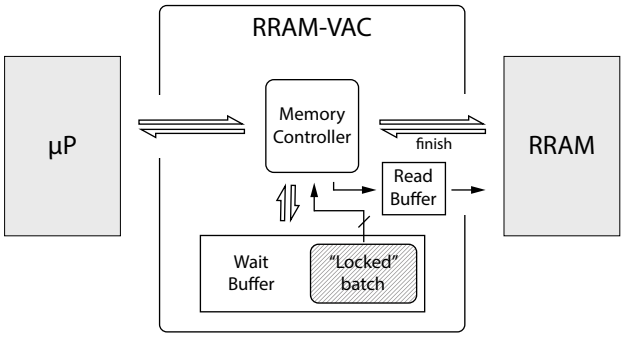

Figure 1: Proposed RRAM-VAC controller block diagram with detailed sub-blocks

3 RRAM-VAC ARCHITECTURE

This section presents the RRAM-VAC architecture, describes its functionality and discusses area and energy considerations.

3.1 Functional Description

In this work, we propose the RRAM Variability-Aware Controller (RRAM-VAC). Figure 1 shows a detailed block diagram of the RRAM-VAC. It relies on the following blocks : (i) a modified memory controller, (ii) a *wait buffer* and (iii) a *read buffer*. The memory controller has two tasks: (i) it routes the memory requests from the processor to the *wait Buffer*, *read buffer* or to the RRAM macro. (ii) Schedule the programming operations from the *wait buffer* to the RRAM memory block. Memory request coming from the processor (*read* or *write*) are stored in the *wait buffer* if they are *write* requests and in the *read buffer* if they are *read* requests. The RRAM-VAC relies on a concept named "Write Coalescing" that we extensively describe Section 3.2.

3.1.1 Write operations. When the *wait buffer* contains enough *write* requests, these requests are locked and considered as what is called a *batch*. Then, the *batch* is written to the RRAM whenever it is filled. The *write* requests from the *batch* are written to the RRAM memory using the Write Coalescing method. As several versions of the same data could be present inside the RRAM-VAC, to satisfy data coherence, we define the following data validity hierarchy: the *wait buffer* contains the latest version of the data. Any request which is not contained inside the *wait buffer* is transferred to the RRAM memory. From a more detailed perspective, while both *read* and *write* requests can be catched by the *wait buffer*, once the *batch* is locked, it can only catch *read* requests. *Write* requests to a locked *batch* have to be considered as new entries inside the *wait buffer*. From that perspective, the RRAM-VAC behaves as a small cache and can avoid sequential access to the same address that would induce an early aging of the memory. It also improves the performances

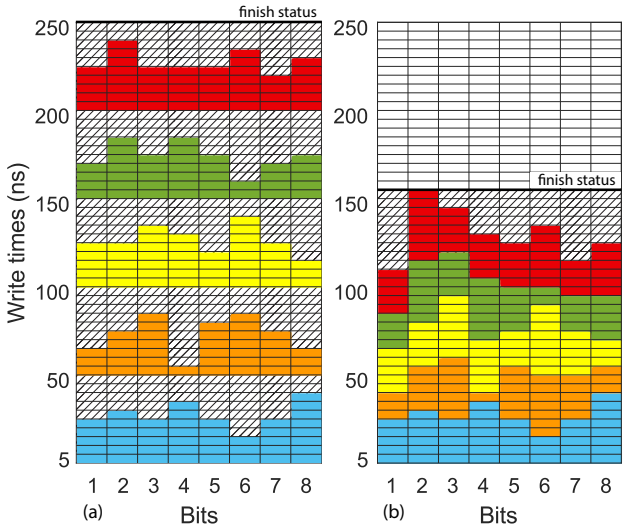

Figure 2: (a) Sequence of 8-bits words written to RRAM in a worst case condition. (b) Same words programmed to RRAM using the proposed RRAM-VAC circuit

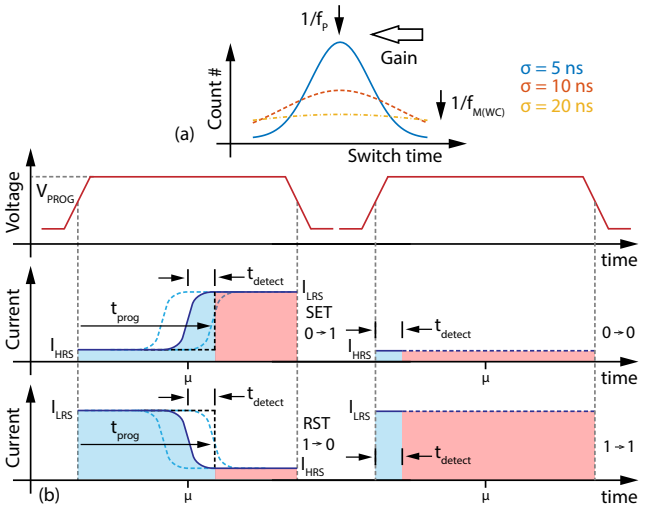

Figure 3: (a) Considered RRAM distributions. (b) Energy calculation in different switching cases and detailed parameters.

and energy consumption of such operations, as *read* and *write* operations to a buffer are faster and less energy hungry than *read* and *write* operations in RRAM. At the end of a computation phase, the *batch* is written to the RRAM even when it is not full.

3.1.2 Read operations. An incoming *read* request is first issued to the *wait buffer*. If the corresponding address is not present, it is issued to the RRAM memory. While the *batch* is written to the RRAM memory, the next *read* request is stored inside the *read buffer* and is performed when the memory is available. During that time, the processor execution is stalled until the RRAM memory is available again. This effect is discussed in Section 5.

3.2 Write Coalescing

Figure 2-a shows the evolution of a sequence of 8-bit words written in memory while considering a worst case programming condition. Colored areas represent the time it actually takes for each of the words to be written to memory (each column represents a bit). Each word is written with a constant programming time accounting for margins and respecting a constant frequency to ensure that, at the next processor cycle, the data is actually written, and the memory is available for the next operation. In these conditions, the hashed area represents the potential energy and performances loss of such an approach. In this context, we propose to coalesce the write operations as shown in Figure 2-b. Every time a bitcell switches, it is detected by the WT circuit and then, the controller issues the next bit write operation to be performed. This way, the overall programming time for the group of words (referred as the *batch* in section 3.1) and the hashed area are drastically reduced. In order to coalesce the words together, several assumptions must be considered. (i) We consider that the words written are interleaved between several arrays, and so, from a sub-array point of view, only one bit is written at a time (this is actually compatible with constraints identified in high density RRAM memories [9]). This way, each sub-array can manage the words independently without having to wait for the entire word to be written. (ii) We consider that the words coalesced are written inside the same memory sub-arrays. In other words, we do not consider parallel write in several sub-arrays that would actually enhance the performances of the RRAM-VAC.

Table 1: Energy corner cases for the considered RRAM

Programming Conditions	With WT		Without WT	
	Slow	Fast	Slow	Fast
Set HRS (i)	0.515pJ	4.515pJ	1.175pJ	4.575pJ
Reset LRS (ii)	0.775pJ	0.17pJ	4.575pJ	1.175pJ
Reset HRS (iii)	0.015pJ		0.75pJ	
Set LRS (iv)	0.1pJ		5pJ	

4 EXPERIMENTAL SETUP

This section introduces the experimental setup considered in this work. First, we present energy considerations regarding the RRAM memory. Then, we present the application characterization methodology used to assess the performances and energy gains provided by the proposed RRAM-VAC circuit.

4.1 RRAM Energy Characterization

In this work, we consider a RRAM technology providing a 50 ns cycle time for the programming operations. As shown in [6], a few tens of ns programming time can be achieved for both *set* (HRS to LRS) and *reset* (LRS to HRS) operations while considering 1 V to 1.5 V programming voltage (V_{prog}). We thereby assume that the programming operations can be performed for the whole distribution at 1V with a 50 ns worst case cycle time. In order to ensure such performances, specific programming strategies such as adaptive

programming voltage [11] can be considered. Finally, we consider a programming current of 100 µA to achieve a sufficient HRS/LRS ratio, a low variability in the LRS state and a several-years retention [6].

As introduced in Section 2.1, cycle-to-cycle and device-to-device temporal variability tends to be extreme (as a reference, in [11], variability may exceed 3 decades). In order to model this effect, we consider a normal distribution on the programming time. Also, we assume balanced programming conditions between *set* (programming operation towards a LRS) and *reset* (programming operation towards a HRS). Figure 3-a shows the three distributions considered: σ = 5 ns, 10 ns and 20 ns. Following the 50 ns programming time, we define a distribution mean (μ = 25 ns) and consider 5 ns margin at the end and at the beginning of the programming pulse, ensuring that all the switching events happen in this window (5 ns to 45 ns). That said, 4 cases may happen : (i) program RRAM devices from one state to the other (HRS to LRS or LRS to HRS). This case, shown in Figure 3-b may induce a high energy consumption without a WT circuit (saved energy, thanks to the WT circuit is represented in red for all these graphs). (ii) program RRAM device in a state where they already are (LRS to LRS or HRS to HRS). In that case, a WT circuit is particularly important as it avoids non-needed programming operations. WT circuit detection time (t_{detect}) is considered 1 ns in the following experiments. As a summary, Table 1 presents the corner programming energy cases. Regarding the *read* performances, we assume that *read* operations can be performed in one cycle. From the energy standpoint, we take as a reference data from [7, 14] and consider a 1 pJ per bit.

4.2 Application Characterization and Simulation Methodology

To assess the gains of the proposed solution, we simulate its functionality with memory traces from real applications. To do so, we tracked the *read* and *write* access to the variables that are kept in memory (i.e. in RRAM) of C-Code applications running on a PC. In this context, we simulated different real applications that are widely used in WBSN Edge devices :

- Data compression algorithm : We considered the Compress sensing (CS) algorithm [4], a 50% lossy compression algorithm used on biosignals before storing them. This application takes as input, a 3 seconds Electrocardiogram (ECG) signal and compresses it.
- Machine learning algorithms : We consider the Epilepsy Seizures detection algorithm from [12]. It contains a Feature Extraction (FE) and a Decision Tree (DT). This application processes a 4 seconds Electroencephalogram signal.
- Specific kernels : We considered two specific kernels widely used in signal processing and machine learning : Matrix Multiplication (MM) and Convolution (Conv) [5, 13]. MM multiplies random 30x30 arrays while Conv convolves random 3x3 and 30x30 arrays.

We then feed the extracted memory traces inside a behavioral model of the RRAM-VAC circuit presented in Section 3. The behavioral model has been implemented using Matlab and simulates the operation of RRAM-VAC while taking as input the memory traces. Energy and area characterization of the RRAM-VAC block are performed based on the proposed implementation from Figure 1. It must be noted, that here, we only ensure that we have pessimistic enough

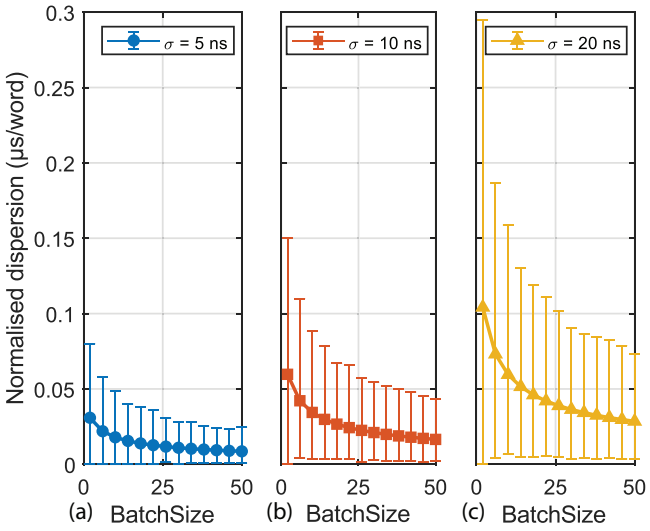

Figure 4: Normalised dispersion in parallel write with different batch sizes

considerations to assess the profitability of the solution. More detailed area and energy considerations are outside the scope of this work and are left for future contributions. From the energy point of view, we assume that for each memory request, a *search* operation is performed in the *wait buffer* address bank. We thereby, took the pessimistic assumption that it is performed inside a BCAM memory [1]. In that context, considering a 0.3fJ/bit/search, would lead from 159 to 192fJ (respectively 64x8-bits and 80x8-bits *wait buffer* address space) per access (it must be noted that values from [1] are extracted from a 128x128array). Conservatively, to ensure worst case estimation, and to account for *read* operations and leakage, we considered 400fJ per access to the RRAM-VAC in our simulations. From the area point of view, assuming a small BCAM and registers would lead to a few thousand μm square. Such area, considering last RRAM published chips [7, 14], corresponds to less than a few equivalent kbits of RRAM and represents less than a percent of area overhead.

5 EXPERIMENTAL RESULTS

In this section, we present (i) a design space exploration of the RRAM-VAC sizing, considering random addresses and data inputs and (ii) a benchmark of the RRAM-VAC while considering various WBSN real application workloads.

5.1 RRAM-VAC Design Space Exploration

Figure 4 presents the normalised dispersion time per *word* versus the *batch* size for a given RRAM-VAC operation frequency situated at the memory programming time average. As introduced in Section 3.2-Figure 2-a, by coalescing the bits, the programming time at word level is averaged. The more bits are written together, the more the variability is averaged. Considering a wider memory time distribution as described Figure 4-a,b and c (that could be explained by a less controlled process, a worn out RRAM device or simply the natural variability of a given technology), leads to

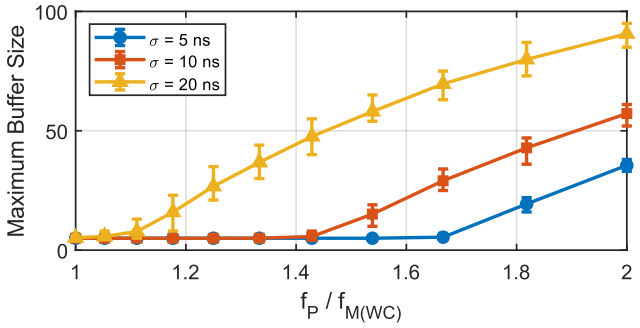

Figure 5: Maximum buffer size requirement with relative processor frequency (in terms of worst case memory frequency) for a batch size of 5

higher distributions as more and more occurrences of slow bits may happen in the *batch*. In that context, considering larger *batch* sizes may help reducing the variability. Alternatively, Figure 5 shows the required *wait buffer* size versus the frequency gain ratio for a *batch* size of 5. In the considered case (introduced in Section 4), the RRAM programming distribution average value is half of the worst case memory frequency ($f_{M(WC)}$), leading to a maximum gain of 2 (beyond that, the *wait buffer* fills faster than it can be flushed to RRAM). As the processor frequency (f_P) increases, the probability of having a series of slow batches (compared to f_P) increases, transiently filling the *wait buffer*. This effect if amplified by the memory variability, as visible for σ = 10 ns and 20 ns. Overall, the sizing of the RRAM-VAC block depends on 3 parameters:

- Memory variability (σ) : high memory variability requires bigger *batch* size.
- Processor Frequency (f_P) : higher f_P requires bigger *wait buffer*.
- *batch* size/*wait buffer* size ratio : A too small *wait buffer* does not average enough the variability and wastes time when the *batch* write finishes in-between two clock ticks. A too big *batch* size forces the system to wait for it to be flushed when the *wait buffer* is full.

Figure 6 summarizes the RRAM-VAC sizing trade-offs. Each contour line corresponds to a sizing for which the processor is never forced to wait while writing random data to random addresses. Figure 6-a shows that for the considered RRAM technology, optimal performances can be achieved with a 75words *wait buffer* and 15words *batch*. Reducing f_P by 10% moves the optimum to the couple 20words/10words, relaxing the area and energy constraints on the RRAM-VAC. On the other hand, Figure 6-b shows that for a constant f_P, a more dispersed memory technology strengthens the constraints and moves the optimum toward bigger *wait buffer* and *batch*. In a looking forward perspective, these trade-offs may open run-time adaptive frequency strategies to take advantage the *wait buffer* and compensate for the RRAM temporal variations.

5.2 Real Workload Exploration

In this subsection, we explore the energy and performances gains provided by the proposed RRAM-VAC concept while running realistic WBSN workloads. The RRAM-VAC is sized accordingly to the

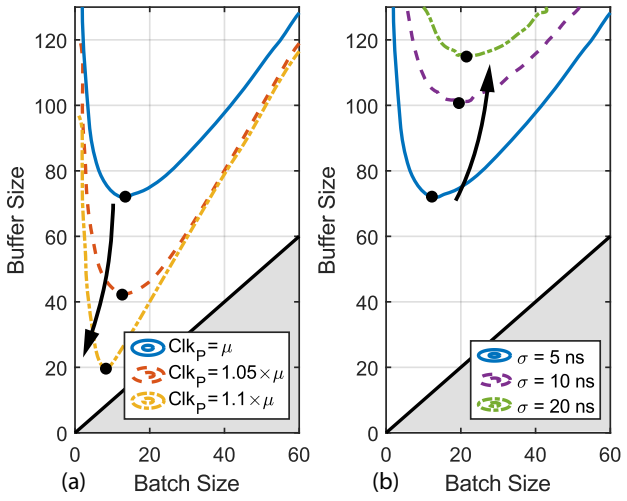

Figure 6: Contour plots showing the optimal sizing of the RRAM-VAC circuit for (a) various f_P and (b) memory temporal variability.

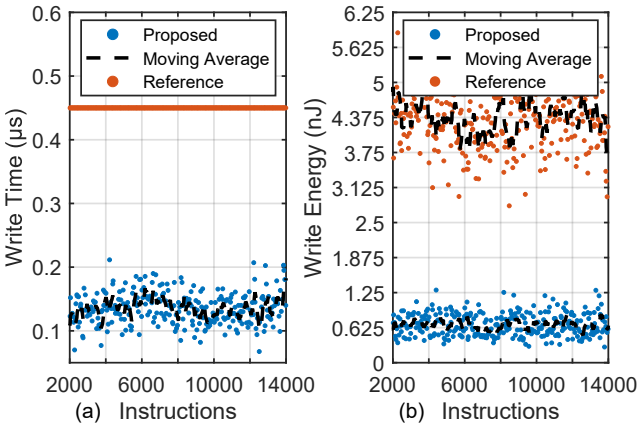

Figure 7: Transient simulations of the RRAM-VAC running the CS application. (a) Performances gains and (b) energy gains per *batch*.

explorations performed in Section 5: *wait buffer* of 80 and *batch* of 10 while the frequency gain is set to $2 \times f_{M(WC)}$.

Figure 7 shows the energy consumption and performance gains of a RRAM memory connected to a processor running the CS application and using the RRAM-VAC (in blue) compared to reference case not using the RRAM-VAC and considering worst case programming conditions (in orange). In these graphs, each dot corresponds to 10 words written (*batch* size). Figure 7-a shows the performances gains enabled by the RRAM-VAC. It shows that compared to the reference system, for the considered memory specifications, the processor frequency can be increased by 2×. On the other hand, Figure 7-b shows a transient simulation of the energy consumed. Thanks to the RRAM-VAC, the energy consumed is reduced compared to the reference case, while running a CS application. Figure

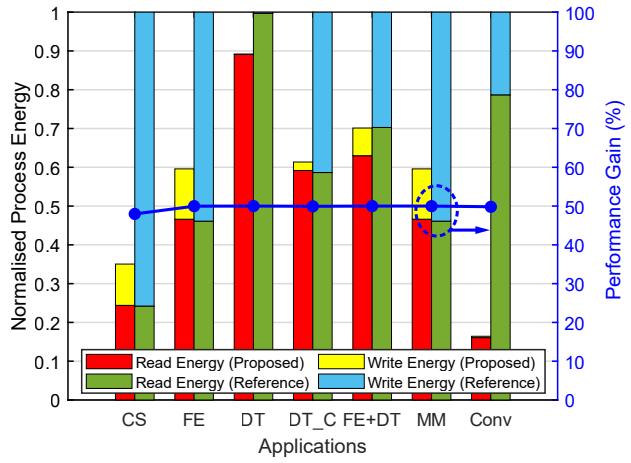

Figure 8: Energy and performances gains of the RRAM-VAC for different applications

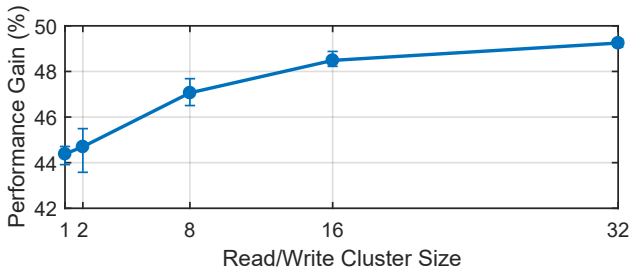

Figure 9: Performance gains of the RRAM-VAC while alternating reads and writes with bursts of different lengths for random inputs (addresses and data) versus burst size.

8 shows the energy and performances gains provided by the RRAM-VAC for the applications presented in Section 4. In terms of energy gains, we separated *write* and *read* energies to ease the understanding. Energy gains provided by the RRAM-VAC appear to be highly dependant on the application memory patterns. Gains on CS, FE and MM applications are highly dependent on the *read*/*write* ratio. On the other hand, DT application shows a 10% performances gain mainly due to the high locality of the *read* operations (i.e., reading a word from the *wait buffer* is less expensive than accessing the RRAM array) and a really low *write/read* ratio (less than 1 per 300). As a reference, we compared it to the DT_C application (DT application memory traces post-processed with a L1 cache of 16 words and least-used eviction policy) and it shows a balanced read/write ratio (as CS, FE and MM) providing stronger gains with the RRAM-VAC. Finally, Conv application shows an even higher amount of *read* and *write* access on recently written words than DT application and exhibits the highest energy gain of all the considered applications, thanks to its high *read* and *write* locality.

In terms of performances, all the considered applications show around 50% of performances improvement compared to the reference case. While the gains are substantial, slight fluctuations can be observed. This is due to the fact that while the *write buffer* is being written to memory, the next *read* request that cannot be processed inside the *wait buffer* is stored inside the *read buffer* and the processor is stalled. This request is finally performed once the memory is available again. In this context, the gains provided by the RRAM-VAC are slightly reduced as the computation is delayed. Figure 9 shows the gains achieved for a random inputs and random addresses application (to avoid any bias provided by the cache effect of the *wait buffer*) versus the *read* and *write* pattern. It shows that longer bursts are less likely to stall the processor than shorter ones. At the worst case, the performance gains are reduced to 44%.

6 CONCLUSION

In this work, we proposed the RRAM Variability Aware Controller (RRAM-VAC), a new controller for RRAM-based memories that takes advantage of the family of Write Termination (WT) circuits to mitigate device-to-device and cycle-to-cycle variability of RRAM technologies. By coalescing the *write* requests and performing them together, it averages the variability and enables strong energy and performances gains. We explored the design space of the RRAM-VAC circuit and estimated its gains by simulating it with memory traces from WBSN applications. With the considered RRAM technology, we show from 44 to 50% performances and 10 to 85% of energy gain depending on the application memory access patterns. Such performance and energy gains makes the RRAM-VAC an extremely promising solution for normally-off Edge devices.

ACKNOWLEDGMENTS

This work has been partially supported by the ERC Consolidator Grant COMPUSAPIEN (GA No. 725657) and by the the ThinkSwiss research scholarship by swissnex India.

REFERENCES

[1] A. Agarwal et al. 2011. A 128x128b high-speed wide-and match-line content addressable memory in 32nm CMOS. In *IEEE ESSCIRC*.
[2] M. Alayan et al. 2019. Switching Event Detection and Self-Termination Programming Circuit for Energy Efficient ReRAM Memory Arrays. *IEEE TCASII*.
[3] D. Apalkov et al. 2016. Magnetoresistive random access memory. *Proc. IEEE*.
[4] J. Constantin et al. 2012. TamaRISC-CS: An ultra-low-power application-specific processor for compressed sensing. In *IEEE/IFIP VLSI-SoC*.
[5] A. Krizhevsky et al. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems 25*.
[6] E. Vianello et al. 2013. Back-End 3D Integration of HfO2-Based RRAMs for Low-Voltage Advanced IC Digital Design. *IEEE ICICDT*.
[7] P. Jain et al. 2019. 13.2 A 3.6Mb 10.1Mb/mm2 Embedded Non-Volatile ReRAM Macro in 22nm FinFET Technology with Adaptive Forming/Set/Reset Schemes Yielding Down to 0.5V with Sensing Time of 5ns at 0.7V. In *IEEE ISSCC*.
[8] A. Lee et al. 2017. A ReRAM-Based NVFF With Self-Write-Termination Scheme for Frequent-OFF Fast-Wake-Up Nonvolatile Processors. *IEEE JSSC*.
[9] A. Levisse et al. 2017. Architecture, design and technology guidelines for cross-point memories. In *IEEE/ACM NANOARCH*.
[10] Panasonic 2018. ReRAM-based MCU MN101L. https://industrial.panasonic.com/ww/products/semiconductors/microcomputers/mn101l
[11] G. Sassine et al. 2018. Sub-pJ consumption and short latency time in RRAM arrays for high endurance applications. In *IEEE IRPS*.
[12] D. Sopic et al. 2018. e-Glass: A Wearable System for Real-Time Detection of Epileptic Seizures. In *IEEE ISCAS*.
[13] A. Vasudevan et al. 2017. Parallel Multi Channel convolution using General Matrix Multiplication. In *IEEE ASAP*.
[14] L. Wei et al. 2019. 13.3 A 7Mb STT-MRAM in 22FFL FinFET Technology with 4ns Read Sensing Time at 0.9V Using Write-Verify-Write Scheme and Offset-Cancellation Sensing Technique. In *IEEE ISSCC*.
[15] H-S Philip Wong et al. 2010. Phase change memory. *Proc. IEEE*.
[16] H-S Philip Wong et al. 2012. Metal–oxide RRAM. *Proc. IEEE*.