OR-ML: Enhancing Reliability for Machine Learning Accelerator with Opportunistic Redundancy

Bo Dong*†, Zheng Wang*, Wenxuan Chen*†, Chao Chen*, Yongkui Yang*, Zhibin Yu*
*Shenzhen Institutes of Advanced Technology, Chinese Academy of Science, Shenzhen, China
†School of Microelectronics, Xidian University, Xi'an, China

***Abstract*—Reliability plays a central role in deep sub-micron and nanometre IC fabrication technology and has recently been reported to be one of the key issues affecting the inference phase of neural networks. State-of-the-art machine learning (ML) accelerators exploit massively computing parallelism observed in neural networks to achieve high energy efficiency. The topology of ML engines' computing fabric, which constitutes large arrays of processing elements (PEs), has been increasing dramatically to incorporate the huge size and heterogeneity of the rapid evolving ML algorithm. However, it is commonly observed that activations of zero value lead to reduced PE utilization. In this work, we present a novel and low-cost approach to enhance the reliability of generic ML accelerators by Opportunistically exploring the chances of runtime Redundancy provided by neighbouring PEs, named as OR-ML. In contrast to conventional redundancy techniques, the proposed technique introduces no additional computing resources, therefore significantly reduces the implementation overhead and achieves obvious level of protection. The design prototype is evaluated using emulated fault injection on FPGA, executing mainstream neural networks for objection classification and detection.**

***Index Terms*—Fault tolerance, machine learning accelerator, opportunistic redundancy**

I. Introduction

The recent advancements in machine learning, especially neural networks (NN), have demonstrated their success in a wide range of application domains, such as computer vision and natural language processing. To accelerate the massive amount of computation in neural networks, domain-specific architectures and circuits have been designed [1] [2], which exploits multiple levels of parallelism to primarily improve energy efficiency. However, one of the critical issues for nanometre IC technology, known as reliability, is usually overlooked for ML accelerators. The logic is that neural networks have the intrinsic capability of accuracy tolerance [3] which has been applied to the approximation method such as network pruning and quantization. However, it is substantially investigated in [4] that the neural network has limited fault tolerance capability without a proper design. In [5] the author quantifies the impact of errors for RTL accelerator, where several injected faults lead to ***disastrous*** effects on MNIST dataset. It is also confirmed in our experiments on ImageNet dataset, present in Section IV, that state-of-the-art neural networks such as Resnet-18 and Yolo-v2 suffers from ***observable degradation*** in accuracy for a PE error probability ranging from 10^{-6} to 10^{-5}, which is close to the soft error rate of SRAM cells in nanometre CMOS technology [6]. Furthermore, [7] demonstrates that both design metrics of energy efficiency and reliability for ML accelerators closely correlate, rendering error mitigation techniques inevitable in the future ML chips.

Fault-tolerant IC design techniques heavily rely on replication for mutual verification of computing resources. However, conventional replication approaches such as triple modular redundancy (TMR) and redundant multithreading (RMT) [8] incur huge hardware overheads. Directly applying replication into ML architecture will exaggerate the overhead issue due to the immense amount of parallel processing elements. In this work, instead of physical duplication of computing resources, ***opportunities*** of replication are dynamically explored depending on the properties of activation values in the processing pipeline. Fault tolerance is achieved by either mutual verification between adjacent PEs with the same activation values or borrowing the neighbouring PE with zero value for double-checking. Consequently, the proposed method enhances reliability with slight physical overhead, while achieves an evident level of protection for mainstream neural networks.

The work is organized as follows. Section II presents the ideal of opportunistic redundancy in ML accelerators. Section III details the architecture and circuit implementation on an exemplary design. Section IV demonstrates the FPGA-based evaluation platform, achieved levels of fault tolerance and implementation overheads. Section V concludes the work.

II. Opportunistic Redundancy in ML Architecture

1) Motivation: A significant amount of zero values exists in the activation datastream, which results from non-linear functions such as ReLU and feature map preprocessing. Besides, spatial locality of feature maps usually causes activations in adjacent streams to be identical. Such cases introduce either invalid or redundant computation, which generates potential threats for processing under a faulty environment. Unprotected bit-flips on either non-zero or zero values can lead to accuracy degradation and even security threat.

Fig. 1 illustrates zero values and adjacently identical values observed in four well-known neural networks across individual convolution layer. The statistics are obtained from Darknet framework [9]. The *opportunity* bars sum up the proportion of identical and zero values, indicating that various networks have distinct distributions of potential protection opportunities. In contrast to three other networks, VGG-16 has huge amounts of zeros, caused by its image preprocessing crop layer [9]. Yolo-v3-tiny on the other hand exhibits sharp increment in identical values on the layer of 1024 channels. Noted that the Darknet framework adopts 32-bit float point (fp32) processing, whereas the majority of ML accelerators processes 8-bit integer (int8), therefore boosts the level of opportunities due to quantized activation values.

This work was supported by national key research and development program under Grant No. 2016YFB1000204, NSFC with Grant No. 61702493, 61902355, Key-Area Research and Development Program of Guangdong Province (Grant No. 2019B010155003), Shenzhen S&T Funding with Grant No. KQJSCX20170731163915914. Bo Dong and Zheng Wang contribute equally to this work. Zheng Wang is the corresponding author (zheng.wang@siat.ac.cn)

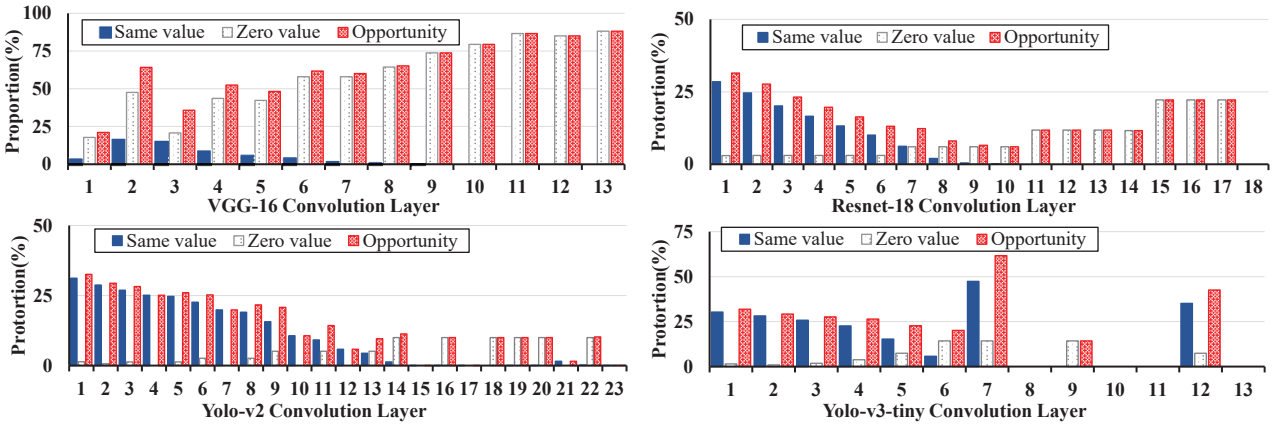

Fig. 1. Statistics of protection opportunities for state-of-the-art neural networks averaged on 100 images of ImageNet dataset

2) Strategy: Mainstream neural network accelerators favour two dimensional PE arrays for dataflow processing [10]. In this work, an exemplary output stationary dataflow accelerator is extended with opportunistic redundancy, while the approach is conveniently customizable for generic dataflow architectures. As present in Fig. 2, the baseline architecture constitutes a 4×4 computing fabric with 16 PEs which processes four datastreams in parallel. Each PE contains at least one multiply-and-accumulate (MAC) unit and accumulates activation value of convolutional and fully connected network layers.

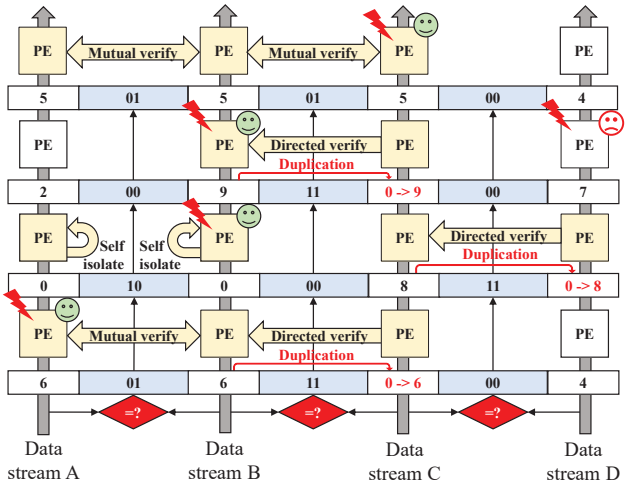

Fig. 2. Scheme of opportunistic redundancy in two dimensional PE array

The proposed technique takes advantage of zero and the identical activation values for computational verification. The protection strategies are illustrated as below:

- *Mutual verify*: Two adjacent PEs with same input activation dynamically form a pair of *mutual verification*, to detect faults on either PE.
- *Directed verify*: A PE with ignorable zero input activation forms a pair of *directed verification* with its neighbouring PE, to detect faults on either PE.
- *Self-isolate*: Two adjacent PEs with both zero activations are *self-isolated* from faulty MAC operation.

To efficiently direct pair forming, adjacent datastreams are compared and encoded as protection vector, passing through the processing pipeline. Once upon fault detection, the processing pipeline is stalled and re-executed. The extended computing fabric opportunistically facilitates redundancy (*OR-ML*) hence chances are explored and exploited whenever they exist, in contrast to duplication of physical resources. Consequently, low-cost and high robustness are simultaneously achieved.

III. Implementation

The implementation of proposed OR-ML architecture with opportunistic redundancy is present in Fig. 3. To form redundant pair dynamically with low overhead, *protection vector*, *chance explore unit* (CEU), *chance taken unit* (CTU) and *re-execution unit* (REU) are introduced and discussed in this section.

1) Protection vector: A two-bit vector encodes the aforementioned protection strategy, as shown in Table I. The vector is ***dynamically*** generated from the outputs of the activation buffers, which requires no offline preprocessing and is regardless of workload sizes. The vectors are propagated through the pipeline of the accelerator along with corresponding datastreams. Parity bits can be employed to enhance the robustness of the vector.

TABLE I
The Interpretation of Protection Vector

MSB	LSB	Protection strategy
0	0	No opportunity
0	1	Mutual verify
1	0	Self-isolate
1	1	Directed verify

2) Chance explore unit: Activation sharing is the fundamental design principle for mainstream ML accelerators. Once upon fetching from the on-chip buffer, the activations are usually propagated through PE arrays to reduce repeated memory access. Instead of *on-site* exploring chances for a huge amount of PEs, the CEU discovers opportunities only for fresh activation streaming out of the buffer. It compares adjacent datastreams to encode and propagate the protection vector through the pipeline. We name the approach as *compare once then propagate (COTP)*. Such an approach provides two advantages: First, a huge additional area caused by on-site comparators is saved. Second, COTP is relatively reliable since the propagated datastreams can easily become

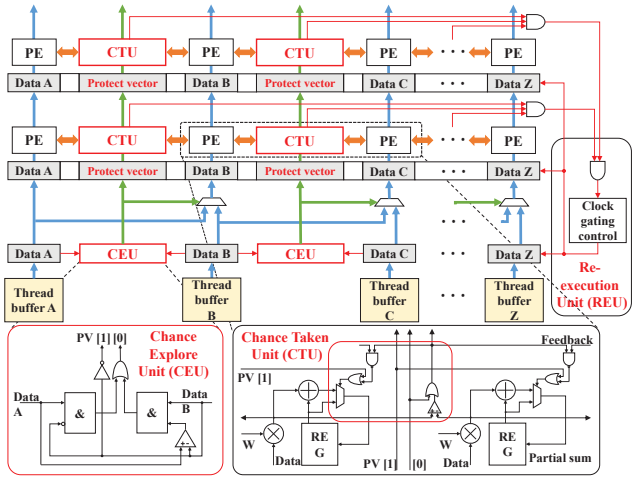

Fig. 3. Implementation of dataflow architecture with opportunistic redundancy

faulty, which mislead the on-site comparators for unintentional behaviours. CEU favours a low-cost design fashion with a few flip-flops and logic gates.

3) Chance taken unit: CTU compares the output values from adjacent PEs for error detection. The LSB of protection vector and the result bit of comparator jointly determine the CTU's control output. Together with the MSB of the protection vector, the control signals determine whether the result of the multiplication is discarded for both strategies of directed and mutual verify. Additionally, zero activation values are by default skipped for PE. However, faulty activation may fool the PE to process it. Therefore under the strategy of self-isolate, CTU directs the PE to skip accumulating the result of MAC, in case that the PE is falsely activated. In terms of physical cost, CTU also favours a low area overhead since only a few 2-input standard cells are necessary.

4) Re-execution unit: REU combines the control outputs generated by CTUs to enable clock gating upon fault detection. The current implementation of REU extends to a baseline output-stationary NN engine [13], where all PEs are executed in a step-locking fashion to minimize control-complexity. Consequently, the gated clock stalls the complete pipeline and re-executes the faulty operation in the next clock cycle. The design methodology is similar to the techniques in [11]. For other microarchitectures, pipeline bubbles can be inserted in specific locations where the errors are detected, where no complete pipeline-stall is required.

IV. Experimental Results

1) Emulated fault injection environment: Although being successfully adopted by reliability research community for years, simulated fault injection experiment using high-level neural network modelling frameworks such as Darknet and PyCharm suffers from two critical issues. First, the scope of injected fault is limited to software variables, which is far from accurate compared with physical faults on ML accelerator with ASIC style design. Second, software simulation in CPU is extremely slow for executing state-of-the-art neural networks with Giga to Tera operations, especially for emulating fixed point algorithmic with floating-point units.

In this work, we construct an emulated fault injection environment in Xilinx Kintex-7 160T FPGA. The FPGA interacts with host PC using Opal Kelly IPs [12] to load bitstream, fault configuration, neural network model, weights and feature maps. The default neural network engine in [13] is augmented with fault injection capability shown in Fig. 4. The fault scheduler reads LFSR seeds from host PC for initializing injection cycles and locations. Upon execution, random numbers are generated on the fly and compared with a threshold value to determine when and where faults should be injected. The input feature maps (Ifmaps) stands for images to be processed by NN. Output feature maps (Ofmaps) with emulated faults are read from FPGA and benchmarked with results of the error-free Ofmaps. The extended design is synthesized under 100MHz frequency using Xilinx Vivado 2018.2.

Regarding the specifications of fault injection campaign, the current faulty locations are limited to the ***output registers of multipliers*** across PEs. For an initial investigation, it is assumed that the CTU, CEU and REU units are error free while further locations will be explored in the extended work. Regarding the type of faults, bit-flip is currently injected to model the transient fault caused by radiations. However, stuck-at-faults in PEs can potentially affect the proposed OR-ML due to propagation of wrong intermediate values. We admit that single protection mechanism cannot cover both bit-flip and stuck-at faults, therefore exploration of combined approaches including but not limited to ramp-up self-scan and dynamic OR-ML can be an adequate solution.

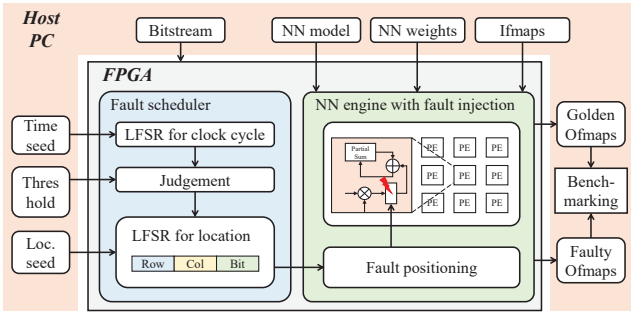

Fig. 4. Fault injection setup on FPGA

2) Effects of protection on ML accelerator: Resnet-18 and YOLO-v2 are executed on the FPGA emulator under fault injection. We introduce ***PE error rate (PER)*** to quantify the percentage of erroneous PE output upon fault injection. It is configured from the FPGA environment as a threshold value which illustrates how many multiplications result in one single fault. For instance, the PER of $1/6 \times 10^{-5}$ stands for 1 fault out of 6×10^{5} multiplications.

Taking Yolo-v2 for instance, as present in Fig. 5, the baseline ML architecture misses class *person* for a PER of 0.5×10^{-6}. However, the OR-ML architecture performs error-free detection for the same PER value while suffers slightly from miss detection when PER rises to 10^{-6}.

To quantify impacts on accuracy, we use TOP1 and TOP5 accuracy for classification networks. For detection network we define detection error as following:

$$DetectionError = \frac{(mAP_{errorfree} - mAP_{actual})}{mAP_{errorfree}} \quad (1)$$

Fig. 5. Fault injection experiment on OR-ML for YOLOv2

Fig. 6 presents classification error with increased PER for Resnet-18 network on OR-ML architecture. The legends OR-Non, OR-Same and OR indicate unprotected design, protect with mutual verify only and protect with all three strategies respectively. 100 images from ImageNet dataset is tested. It is observed that OR-ML with mutual verify strategy reduces classification errors on both TOP1 and TOP5 metrics compared to the baseline architecture. The directed verify and self-isolate strategies also contribute to enhancing reliability.

On the contrary, Fig. 7 presents detection error with increased PER for Yolo-v2 network with similar benchmarking methodology. The result indicates the trend of reliability enhancement for the proposed strategies.

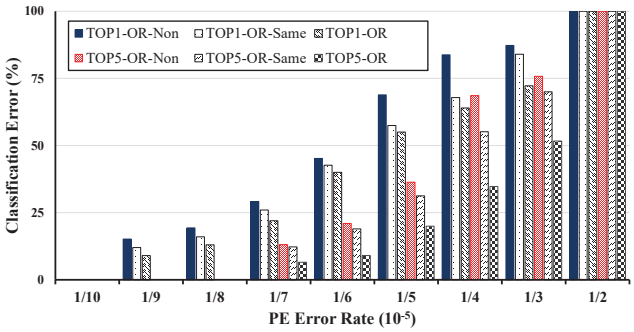

Fig. 6. Classification errors on baseline and OR-ML for Resnet-18

3) Physical overheads: The OR-ML architecture with 512 PEs (organized in 16 columns and 32 rows) is completely designed in Verilog and synthesized by 40nm SMIC library under 200MHz using Synopsys Design Compiler. As present in Tab. II, the OR-ML accelerator introduces only 3.55% area overhead, mainly caused by CEU, CTU and pipeline registers for protection vector. 26.98% power overhead is estimated mostly due to additional sequential logic.

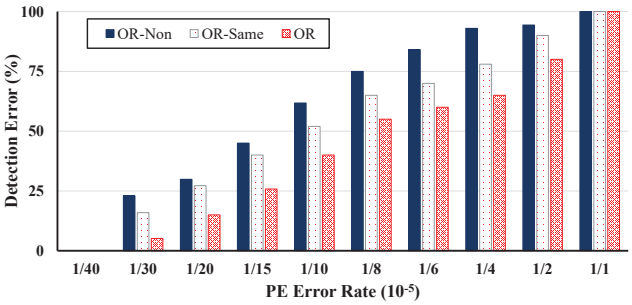

Fig. 7. Detection errors on baseline and OR-ML for Yolo-v2

TABLE II
PHYSICAL ESTIMATES OF BASELINE AND OR-ML ACCELERATORS

SMIC 40nm tech.	Area (μm^2)	Power (mW)	Frequency (MHz)
Baseline-ML	3262786	50.2012	200
OR-ML	3378564	63.7463	200

V. CONCLUSION

This work presents reliability enhancement techniques for ML accelerator exploiting opportunistic redundancy. Compared with conventional hardware replication techniques, the proposed OR-ML architecture dynamically explores chances for replication without incorporating extra computing resources. Several protection strategies are proposed with a few amendments to an exemplary ML engine with output stationary dataflow. The effect of fault tolerance is demonstrated on mainstream neural networks using the emulated FPGA platform with fault injection capability. Implementation with 40nm SMIC technology shows that only 3.55% area overhead is incurred compared to the baseline ML engine.

REFERENCES

[1] Y. Chen *et al.*, "Dadiannao: A machine-learning supercomputer," in *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 609–622, IEEE Computer Society, 2014.
[2] Y.-H. Chen *et al.*, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE Journal of Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, 2017.
[3] H. Esmaeilzadeh *et al.*, "Neural acceleration for general-purpose approximate programs," *IEEE Micro*, vol. 33, no. 3, pp. 16–27, 2013.
[4] C. Torres-Huitzil and B. Girau, "Fault and error tolerance in neural networks: A review," *IEEE Access*, vol. PP, pp. 1–1, 2017.
[5] B. Salami, O. S. Unsal, and A. C. Kestelman, "On the resilience of rtl nn accelerators: Fault characterization and mitigation," in *2018 30th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, pp. 322–329, IEEE, 2018.
[6] C. Slayman, "Soft errors—past history and recent discoveries," in *2010 IEEE International Integrated Reliability Workshop Final Report*, pp. 25–30, IEEE, 2010.
[7] B. Reagen *et al.*, "Minerva: Enabling low-power, highly-accurate deep neural network accelerators," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, 2016.
[8] S. K. Reinhardt and S. S. Mukherjee, "Transient fault detection via simultaneous multithreading," in *Computer Architecture, 2000. Proceedings of the 27th International Symposium on*, 2000.
[9] J. Redmon, "Darknet: Open source neural networks in c." http://pjreddie.com/darknet/, 2013–2016.
[10] V. Sze *et al.*, "Efficient processing of deep neural networks: A tutorial and survey," *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017.
[11] D. Ernst, N. S. Kim, S. Das, S. Pant, R. Rao, T. Pham, C. Ziesler, D. Blaauw, T. Austin, K. Flautner, *et al.*, "Razor: A low-power pipeline based on circuit-level timing speculation," in *Proceedings. 36th Annual IEEE/ACM International Symposium on Microarchitecture, 2003. MICRO-36.*, pp. 7–18, IEEE, 2003.
[12] O. Kelly. https://opalkelly.com/, 2013–2016.
[13] Z. Wang, L. Zhou, W. Xie, W. Chen, J. Su, W. Chen, A. Du, S. Li, M. Liang, Y. Lin, *et al.*, "Accelerating hybrid and compact neural networks targeting perception and control domains with coarse-grained dataflow reconfiguration," *Journal of Semiconductors*, vol. 41, no. 2, p. 022401, 2020.