Impact of Thread Parallelism on the Soft Error Reliability of Convolution Neural Networks

Geancarlo Abich*, Rafael Garibotti†, Jonas Gava*, Ricardo Reis*, Luciano Ost‡
* PGMicro/UFRGS, Universidade Federal do Rio Grande do Sul, Porto Alegre, Brazil – {*gabich,jfgava,reis*}*@inf.ufrgs.br*
† School of Technology, Pontifical Catholic University of Rio Grande do Sul, Brazil – *rafael.garibotti@pucrs.br*
‡ Wolfson School, Loughborough University, United Kingdom – *l.ost@lboro.ac.uk*

2022 IEEE 13th Latin America Symposium on Circuits and System (LASCAS) | 978-1-6654-2008-2/22/$31.00 ©2022 IEEE | DOI: 10.1109/LASCAS53948.2022.9789088

***Abstract*—Convolution neural networks (CNNs) have been incorporated into resource-constrained edge devices to intelligently manage and process local data coming from a variety of sensors. Thread parallelism has been used to boost the performance of neural networks, but only few works address the effect of these parallel modifications on the soft error reliability of underlying models running on edge devices. In this sense, this work aims to assess the soft error reliability of a multi-threaded version of a CNN model developed based on the Arm CMSIS-NN kernels. Results show that the developed threaded CNN model increases performance at the cost of low memory footprint overhead. Promoted multi-threaded CNN model also provides better soft error reliability w.r.t. the original sequential version.**

***Index Terms*—Thread Parallelism, Soft Error Reliability, Machine Learning, Edge Devices.**

I. Introduction

The rising of edge computing systems requires that even resource-bound devices must be able to efficiently predict and recognize patterns of how systems could handle complex environments and unexpected situations [1]. This has generated a movement among researchers and industry to provide solutions that allow the execution of bespoke and optimised machine learning (ML) algorithms into resource-constrained edge devices [2]. For example, software libraries and quantization techniques have been proposed to reduce traditional models' memory and computational requirements. These approaches enable the efficient execution of ML algorithms, mostly CNN models, on low energy and reduced memory footprint platforms.

Recent works have been proposed parallel solutions to lift the performance of neural network models [3], [4]. Garofalo *et al.* [3] provide quantized neural network kernels ranging from 8-bit to 1-bit fixed-point integer inferences targeting a multi-core RISC-V based cluster. Tabanelli *et al.* [4] employ non-neural ML kernels to maximize the speedup while using floating-point emulations on a multi-core RISC-V platform.

Although a wide range of approaches enables the execution of ML algorithms on edge devices, software engineers must consider the reliability while developing lightweight and performance-efficient software to avoid catastrophes in safety-critical applications. For example, Abich *et al.* [5] showed that ML algorithms' reliability executed on resource-constrained devices depends on the instruction set architecture and the layer where the faults are injected. In addition, several works have been conducted to assess the reliability of ML techniques considering radiation-induced soft errors [6], mixed-precision functions [7], hardware accelerators [8], software frameworks [9], GPUs [10], FPGAs [11], and virtual platforms [12]. While most of the existing works consider HPC or architecture-specific approaches [8]–[11], the works [10] and [12] demonstrate that the parallelization of ML algorithms affects their soft error reliability.

In this regard, the *contribution* of this paper is twofold. First, the proposal of a parallel version of the specialised industrial CMSIS-NN kernel [13] considering SIMD (i.e., single instruction multiple data) and Thumb instruction sets. This work employs the Open Multi-Processing (OpenMP) [14] application programming interface (API) to enable multi-threaded capacity on CMSIS-NN kernels aiming at executing a CNN trained on CIFAR-10 dataset in an Arm Cortex-A7 processor considering single, dual, and quad-core configurations. Second, the soft error reliability assessment of the proposed parallel model when varying the instruction set and the number of threads.

II. Fault Injection Framework

This work adopts the SOFIA framework [15], an open-source toolset that provides several facilities and well-accepted single-bit-upsets fault injection (FI) techniques. For example, the SOFIA framework can inject faults into physical memory and register file, as well as other places related to the application or architecture structure. To assess the soft error reliability, this work uses the *random register file* FI technique, a well-accepted fault injection that similarly stimulates all general-purpose registers running application and operating system codes.

Unlike traditional fault classification methods supported in SOFIA, the adopted fault classification consists of four different classes that identify the impact of a failure on the CNN output prediction. The first class, called *correct outputs*, are related to those where the output data is the same as expected, i.e., equal to the gold reference data. The second class is the *tolerable faults*, where the output classification remains the same as the faultless. However, the output data does not match the gold reference data. The third and most worrying class is the *critical faults* because the fault classification is incorrect or even not classified, leading to catastrophic events in safety-critical applications. Lastly, *crash* is the class where the system receives an error indication, either because it ended abnormally

or took longer than required, which causes it to be removed from the system.

To provide a proper soft error reliability analysis, this work relies on the *mean work to failure* (*MWTF*) metric [16]. The MWTF relates the application's runtime to the most critical architecture vulnerability to reveal the average amount of work an application can perform before it hits a failure.

The MWTF formula is shown in Equation 1, where Architecture Vulnerability Factor (AVF) corresponds to the chance that a fault will turn into an error [17], e.g., in the case of silent data corruption. In this regard, this work uses the most critical architecture vulnerability factor ($AVF_{CriticalFaults}$), which is related to *critical faults*, where a wrong classification can be catastrophic. For example, in an autonomous vehicle, not detecting a person or cyclist can cause human loss.

$$MWTF = \frac{1}{(execution\ time \times AVF_{CriticalFaults})} \quad (1)$$

III. Case Study: CNN based on CMSIS-NN kernels

This work implemented a multi-threaded version of the convolution neural network based on the CMSIS-NN kernel [13] applying to the Arm Cortex-A processor (i.e., ARMv7-A). The CNN application was trained with the CIFAR-10 dataset [18]. This dataset has 60,000 32x32 colour images and ten output classes. The CNN application comprises repetitive layers composed of convolution, Rectified Linear Unit (ReLU) activation, and max-pooling layers added at the end to a fully-connected layer.

The CMSIS-NN kernel code is composed of two types of functions. The first is related to the functions that implement popular neural network layer types (*NNFunctions*), and the second includes the utility functions (*NNSupportFunctions*) [13]. Unlike NN models trained with 32-bit floating-point data, the CMSIS-NN kernels deploy optimizations to reduce memory footprint using low-precision fixed-point integer representation. These optimizations target Cortex-M systems with SIMD instructions support since such instructions are widely used in NN models, especially 16-bit Multiply-and-Accumulate (MAC) instructions. Furthermore, according to Lai *et al.* [13], these NN models achieve good performance by applying accumulation with dedicated SIMD MAC instruction and using data transformation without reordering.

The first experiment analyses the CNN profile to understand the CNN application structure and which functions are most used. Table I shows the execution time percentages for a single thread running on the Arm Cortex-A7 processor according to each CNN layer kernel. These values demonstrate that the convolution layers execute almost all the time (i.e., 97%) in both the Thumb and SIMD instruction sets. Therefore, these layers are the best candidates to be parallelized to achieve such a performance improvement. In this regard, the following Section describes changes made to the CMSIS-NN kernel to allow parallel execution of convolution layers.

TABLE I
CNN runtime percentages according to layer type

Layer Type	Thumb Runtime (%)	SIMD Runtime (%)
Convolution	97.15	97.28
MaxPooling	2.59	2.22
ReLU	0.23	0.40
Fully-Connected	0.02	0.10

A. Parallel Convolution Kernels

To improve the performance of computational and memory limited edge devices, current approaches have been applying fixed-point arithmetic and quantization of weights and activations on 8-bit [13] or smaller data types [2]. On the other hand, this work extends the CMSIS-NN convolution kernel to support multiple threads considering the SIMD and Thumb instruction sets when executed on an Arm Cortex-A7 processor. The proposed extension relies on the OpenMP library [14] to support threads according to the number of cores available in the adopted processor. The main feature of this extension is that it does not change the goals of the CMSIS-NN kernel, i.e., it keeps the performance and memory optimizations with a low memory footprint overhead ($< 1\%$). Figure 1 demonstrates an example of pseudo-code where the proposed extension instantiates the threads in the convolution kernel.

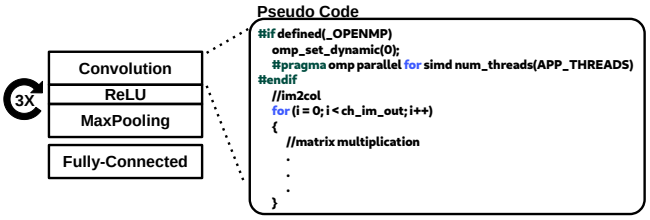

Fig. 1. CMSIS-NN extension to provide a parallel convolution kernel.

The convolution kernel usually comprises two operations. First, the input reordering and expansion. Then, matrix multiplication operations are applied. In this regard, threads are defined between the two operations not to change the CMSIS-NN kernel's characteristics. Furthermore, the *APP_THREADS* environment variable specifies the maximum number of threads to run in parallel, and the *omp_set_dynamic* method guarantees that the number of threads will be the same as defined by the user in the *num_threads* method, avoiding bottlenecks, as shown in Figure 1. In addition, the *simd* and *ordered* directives indicate that the loop can be turned into a SIMD loop (i.e., multiple iterations of the loop can be executed concurrently using SIMD instructions), and those operations must be ordered to ensure they do not affect the inference precision.

IV. Multi-thread Speedup Analysis

The first experiment aims to reveal the speedup gains when applying thread parallelism to the CMSIS-NN convo-

lution kernel. Table II shows the experimental setup used to evaluate the relative speedup considering the Thumb and SIMD instruction sets. To ensure results consistency, we consider the GCC compiler version 9.3 along with optimization flag $-O2$ and OpenMP version 4.5 for all experiments. The optimizations from the CMSIS-NN library are disable in Thumb versions by the architecture specific flag $-D\_\_ARM\_FEATURE\_DSP = 0$, ensuring that the object code is generated entirely by the compiler. To provide accurate results regarding application runtime, we validate the proposed CMSIS-NN extension on a Raspberry Pi 2 (RPi2) Model b board [19]. In this sense, the number of threads is defined ranging from 1 to 4, since the RPi2 board is composed of a quad-core Arm Cortex-A7 processor.

TABLE II
VALIDATION EXPERIMENTAL SETUP

Evaluation Board	Raspberry Pi 2 Model b
Processor	Arm Cortex-A7
ML Model	Cifar-10 CNN
Software Stack	CMSIS-NN
Dataset	CIFAR-10
Compiler and optimization flag	GCC 9.3 with $-O2$
OpenMP version	4.5
Instruction Sets	Thumb and SIMD
Parallel Threads	1, 2, 3, and 4

Figure 2 shows the speedup improvement applying multi-threaded parallelism in the adopted CNN application. Such results are calculated from the execution time obtained from the multi-thread execution over the single thread version. The SIMD-based CNN application presents a low speedup improvement in multi-threaded versions. On the other hand, the Thumb-based CNN application shows an almost linear speedup improvement (i.e., up to 3.5× improvement w.r.t. single-core version). This performance difference is because the Thumb implementation (i.e., generic convolution) has no optimization, whereas the SIMD version promotes architecture-specific CMSIS-NN optimizations. In this sense, the compiler can optimize the object code of the Thumb instruction set more than SIMD. However, the SIMD vs Thumb curve shows that SIMD still has a performance 2.8× higher than Thumb even with the multi-core speedup.

V. SOFT ERROR RELIABILITY ANALYSIS

This Section explores the system soft error reliability by injecting faults (i.e., flipped bits) on processor registers during the execution of the CNN application with the proposed CMSIS-NN extension.

A. Experimental Setup

To assess the reliability of the proposed CMSIS-NN extension, we have used the SOFIA framework to inject faults in the general-purpose registers of the Arm Cortex-A7 processor

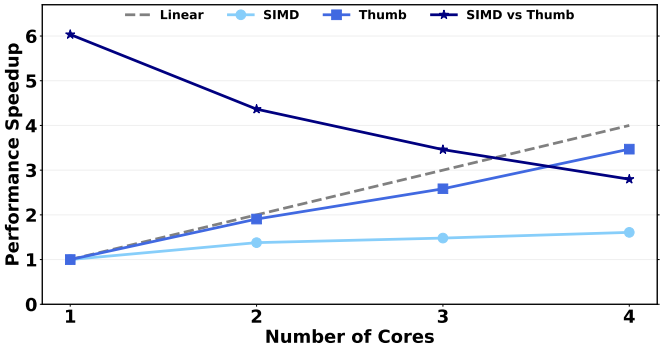

Fig. 2. Speedup results for 1, 2, 3 and 4 thread implementations for Thumb and SIMD instruction sets.

(i.e., r0-r15). Note that all CMSIS-NN optimizations, including SIMD instructions and OpenMP extension, only require general-purpose registers. Table III shows the FI experimental setup, which includes experiments with Thumb and SIMD instruction sets considering 1, 2, and 4 cores/threads running on the Arm Cortex-A7 model supported by the SOFIA framework. Such experimental setup considers the same version of compiler and optimization flags that was used for speedup analysis, keeping the consistency of the FI campaign results.

TABLE III
FI EXPERIMENTAL SETUP

Processor	Arm Cortex-A7
ML Model	Cifar-10 CNN
Software Stack	CMSIS-NN
Dataset	CIFAR-10
Compiler and optimization flag	GCC 9.3 with $-O2$
OpenMP version	4.5
Instruction Sets	Thumb and SIMD
Target FI	General-purpose registers
Number of FI campaigns	6
Injections per campaign	17k
Total Fault Injections	102k

Note that each FI campaign contains N experiments, where each one contains a single input image and one CNN execution. Therefore, faults cannot be propagated to subsequent CNN executions. In addition, developing a realistic and precise approach is one of the main concerns for assessing the soft error reliability of a system. In this sense, this work applies the equations developed by Leveugle *et al.* [20]. These equations ensure the statistical significance of the fault injection results. Thus, each FI campaign has 17000 experiments to generate results with a 1% error margin and 99% confidence level.

B. Soft Error Reliability Assessment

Figure 3 presents the fault injection campaign results executing the adopted CNN application considering Thumb and SIMD instruction sets. The x-axis shows the Arm Cortex-A7

configurations for single, dual, and quad-core systems. The left y-axis shows the fault classes, while the red dots on the right y-axis show the MWTF gain compared to the single thread execution version.

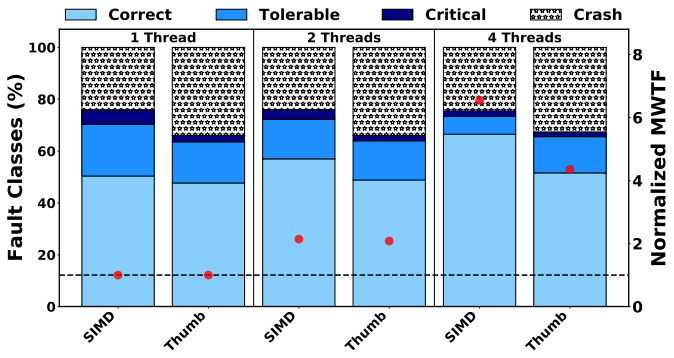

Fig. 3. Results showing fault classifications for the thread parallelism considering single, dual, and quad-core configurations.

The results show that soft errors can affect the Thumb version more than the optimised SIMD version in terms of crashes. These faults are mostly unexpected terminations (e.g., handling exceptions) and are faults detected by the architecture. Even when the number of threads is compared, the same behaviour is observed (i.e., similar percentages in single, dual, and quad-core). This is mainly due to a large number of control and memory instructions between the calculations. Furthermore, the CNN application runs on a Linux kernel; thus, the longer the execution time, the more kernel control instructions are executed.

On the other hand, the SIMD version exploits MAC instructions to provide optimizations in the inference phase, reducing the fault impact. In addition, CNN's performance improvement by increasing the number of threads also increases correct outputs and tolerable faults. However, the SIMD instruction set presented a slight increase in critical faults. This behaviour is caused by the more significant number of SIMD MAC instructions executed between load/store operations, which are more sensitive to soft errors affecting the registers that contain the inference data.

Finally, Figure 3 also shows the impact of soft error reliability on parallel applications. Regarding the execution on dual and quad-core Arm Cortex-A7 processors, we can see that increasing the number of threads positively impacts the soft error reliability, reducing affected outputs (i.e., tolerable and critical faults) and increasing correct outputs for both SIMD and Thumb versions. Although the SIMD version shows a higher percentage of critical faults, it has the best trade-off between reliability and performance when comparing the normalised MWTF. This is due to the architecture-specific optimizations of CMSIS-NN kernels aimed at SIMD instructions. According to the results shown in Figure 2, even with an almost linear speedup in Thumb parallel execution, the SIMD parallel version still provides $3\times$ more performance than the Thumb, consequently impacting the resulting MWTF.

VI. Conclusions

This paper assesses the soft error reliability of a CIFAR-10 trained CNN application, which was developed based on the CMSIS-NN kernel to support multi-threaded execution. The proposed extension showed that threaded parallelism could increase the performance of the adopted CNN application with a low memory footprint overhead (i.e., less than 1%). Furthermore, the results showed that the evaluated CNN application is highly susceptible to failures as it presents critical faults in both configurations. In turn, multi-threaded versions positively impact CNN reliability while increasing the number of correct outputs, performance and, consequently, the MWTF of the adopted CNN.

References

[1] M. S. Mahdavinejad *et al.*, "Machine learning for Internet of Things data analysis: A survey," *Digital Communications and Networks*, vol. 4, no. 3, pp. 161–175, 2018.

[2] A. Capotondi *et al.*, "CMix-NN: Mixed Low-Precision CNN Library for Memory-Constrained Edge Devices," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 67, no. 5, pp. 871–875, 2020.

[3] A. Garofalo *et al.*, "PULP-NN: Accelerating Quantized Neural Networks on Parallel Ultra-Low-Power RISC-V Processors," *Philosophical Transactions of the Royal Society A*, vol. 378, no. 2164, pp. 1–21, 2020.

[4] E. Tabanelli *et al.*, "DNN is not all you need: Parallelizing Non-Neural ML Algorithms on Ultra-Low-Power IoT Processors," 2021. [Online]. Available: https://arxiv.org/abs/2107.09448

[5] G. Abich *et al.*, "Soft Error Reliability Assessment of Neural Networks on Resource-constrained IoT Devices," in *ICECS*, 2020, pp. 1–4.

[6] M. G. Trindade *et al.*, "Assessment of Machine Learning Algorithms for Near-Sensor Computing under Radiation Soft Errors," in *ICECS*, 2020, pp. 1–4.

[7] G. Abich *et al.*, "Applying Lightweight Soft Error Mitigation Techniques to Embedded Mixed Precision Deep Neural Networks," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 68, pp. 4772–4782, 2021.

[8] G. Li *et al.*, "Understanding Error Propagation in Deep Learning Neural Network (DNN) Accelerators and Applications," in *SC*, 2017, pp. 1–12.

[9] B. Reagen *et al.*, "Ares: A framework for quantifying the resilience of deep neural networks," in *DAC*, 2018, pp. 1–6.

[10] F. F. dos Santos *et al.*, "Analyzing and Increasing the Reliability of Convolutional Neural Networks on GPUs," *IEEE Transactions on Reliability*, vol. 68, no. 2, pp. 663–677, 2018.

[11] N. Khoshavi, C. Broyles, and Y. Bi, "A Survey on Impact of Transient Faults on BNN Inference Accelerators," 2020. [Online]. Available: https://arxiv.org/abs/2004.05915

[12] F. R. da Rosa *et al.*, "Using Machine Learning Techniques to Evaluate Multicore Soft Error Reliability," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 66, no. 6, pp. 2151–2164, 2019.

[13] L. Lai *et al.*, "CMSIS-NN: Efficient Neural Network Kernels for Arm Cortex-M CPUs," 2018. [Online]. Available: https://arxiv.org/abs/1801.06601

[14] L. Dagum and R. Menon, "OpenMP: an industry standard API for shared-memory programming," *IEEE Computational Science and Engineering*, vol. 5, no. 1, pp. 46–55, 1998.

[15] V. Bandeira *et al.*, "Non-intrusive Fault Injection Techniques for Efficient Soft Error Vulnerability Analysis," in *VLSI-SoC*, 2019, pp. 123–128.

[16] G. A. Reis *et al.*, "Software-controlled fault tolerance," *ACM Transactions on Architecture and Code Optimization*, vol. 2, no. 4, pp. 366–396, 2005.

[17] S. S. Mukherjee *et al.*, "A Systematic Methodology to Compute the Architectural Vulnerability Factors for a High-Performance Microprocessor," in *MICRO*, 2003, pp. 29–40.

[18] A. Krizhevsky, "Learning Multiple Layers of Features from Tiny Images," University Toronto, Tech. Rep., 2009.

[19] R. Pi, "Raspberry Pi 2 Model b," 2021. [Online]. Available: https://www.raspberrypi.org

[20] R. Leveugle *et al.*, "Statistical Fault Injection: Quantified Error and Confidence," in *DATE*, 2009, pp. 502–506.