Exploring Hardware Fault Impacts on Different Real Number Representations of the Structural Resilience of TCUs in GPUs

electronics

MDPI

Article

Exploring Hardware Fault Impacts on Different Real Number Representations of the Structural Resilience of TCUs in GPUs †

Robert Limas Sierra *, Juan-David Guerrero-Balaguera, Josie E. Rodriguez Condia and Matteo Sonza Reorda

Department of Control and Computer Engineering (DAUIN), Politecnico di Torino, 10129 Torino, Italy; juan.guerrero@polito.it (J.-D.G.-B.); josie.rodriguez@polito.it (J.E.R.C.); matteo.sonzareorda@polito.it (M.S.R.)

* Correspondence: robert.limassierra@polito.it

† This paper is an extended version of our paper published in 31st IEEE Conference on Very Large-Scale Integration (VLSI-SoC) 2023, Sharjah, United Arab Emirates, 15–18 October 2023.

Abstract: The most recent generations of graphics processing units (GPUs) boost the execution of convolutional operations required by machine learning applications by resorting to specialized and efficient in-chip accelerators (Tensor Core Units or TCUs) that operate on matrix multiplication tiles. Unfortunately, modern cutting-edge semiconductor technologies are increasingly prone to hardware defects, and the trend to highly stress TCUs during the execution of safety-critical and high-performance computing (HPC) applications increases the likelihood of TCUs producing different kinds of failures. In fact, the intrinsic resiliency to hardware faults of arithmetic units plays a crucial role in safety-critical applications using GPUs (e.g., in automotive, space, and autonomous robotics). Recently, new arithmetic formats have been proposed, particularly those suited to neural network execution. However, the reliability characterization of TCUs supporting different arithmetic formats was still lacking. In this work, we quantitatively assessed the impact of hardware faults in TCU structures while employing two distinct formats (floating-point and posit) and using two different configurations (16 and 32 bits) to represent real numbers. For the experimental evaluation, we resorted to an architectural description of a TCU core (PyOpenTCU) and performed 120 fault simulation campaigns, injecting around 200,000 faults per campaign and requiring around 32 days of computation. Our results demonstrate that the posit format of TCUs is less affected by faults than the floating-point one (by up to three orders of magnitude for 16 bits and up to twenty orders for 32 bits). We also identified the most sensible fault locations (i.e., those that produce the largest errors), thus paving the way to adopting smart hardening solutions.

Keywords: floating-point numbers; graphics processing units (GPUs); permanent faults; posit numbers; real number arithmetic; Tensor Core Units (TCUs)

Citation: Limas Sierra, R.; Guerrero-Balaguera, J.-D.; Condia, J.E.R.; Sonza Reorda, M. Exploring Hardware Fault Impacts on Different Real Number Representations of the Structural Resilience of TCUs in GPUs. *Electronics* **2024**, *13*, 578. https://doi.org/10.3390/electronics13030578

Academic Editors: Savvas A. Chatzichristofis, Minas Dasygenis and Athanasios Kakarountas

Received: 22 December 2023
Revised: 26 January 2024
Accepted: 29 January 2024
Published: 31 January 2024

1. Introduction

Currently, hardware accelerators are essential units that allow the achievement of the high computational power demanded for a broad number of modern systems, ranging from mobile to high-performance computing (HPC) applications [1,2]. Most modern applications (e.g., those using machine learning algorithms) rely on linear algebra operations, particularly *general matrix multiplication* (GEMM or M × M). In fact, algorithms for M × M are intrinsically parallel and can be optimized by resorting to specialized hardware topologies, such as *systolic arrays* (SAs) [3], *dot product units* (DPUs) [4], and *in-memory computing* [1].

These topologies are typically incorporated as main engines in application-specific devices like Tensor Processing Units (TPUs) in the artificial intelligence (AI) domain. At the same time, graphic processing units (GPUs) evolved as powerful, popular, and flexible

accelerators in the market, and current generations incorporate dedicated and specialized in-chip hardware accelerators for M × M operations (*matrix cores processing units* and Tensor Core Units or TCUs) to speed up performance by providing mixed-precision computations [5].

The computational flexibility of TCUs makes them very suitable for all applications based on AI, such as scientific computing, cryptography [6], image and video processing [5,7], virtual reality [8], the Internet of Things (IoT) and the Internet of Multimedia Things (IoMT) [9], wireless communication [10], multidimensional processing data [11–18] , and complex environment data processing [19]. In fact, several works have proposed methods to increase the accuracy and performance of parallel algorithms using tensor-based models [11]. Out of all these application domains, in this paper, we specifically focus on the reliability aspects of safety-critical applications (e.g., autonomous driving, robotics, space, and healthcare systems), which are nowadays pervaded with GPU-based devices. In all these systems, TCUs play a fundamental role in conducting advanced and autonomous tasks [20]. Moreover, the reliability of safety-critical systems represents a paramount aspect that has to be accomplished by satisfying strict safety standards (e.g., ISO26262 [21] in the automotive domain [22]). Hence, the TCUs inside GPUs must fulfill those reliability criteria as well.

Unfortunately, the astonishing performance of in-chip TCUs within GPUs can be overshadowed by the reliability concerns associated with the vulnerabilities of modern semiconductor technologies that make them prone to faults [23,24]. Permanent faults model possible defects that can happen due to different situations, such as (i) test-escaped manufacturing flaws, (ii) infant mortality phenomena, and (iii) abrupt damage to the device during in-field operations caused by process variation, premature aging, harsh environments, or high operating temperatures [23]. Furthermore, during the in-field operation of a GPU, permanent faults in TCUs can be eventually activated, and their effects propagate silently to the outputs, leading to catastrophic consequences that endanger the reliability and overall safety of a running application. In this regard, evaluating the impact of permanent faults in TCUs is imperative to identify critical hardware vulnerabilities and devise suitable hardening strategies.

So far, most of the efforts to evaluate the reliability of GEMM accelerators have concentrated on assessing the effect of transient faults on different SA topologies [25–27]. The authors of [28] analyzed the impact of soft errors on machine learning accelerators (e.g., NVDLA). Their results show that most of the fault effects might produce small variations during the training/inference stages of CNNs (i.e., up to 90.3% of all effects do not significantly affect the expected result). However, their evaluations were limited to soft error effects on CNN workloads. In [29], the authors evaluated the influence of radiation effects in TCUs and mixed-precision formats of GPUs on the overall reliability of M × M operations. The reported experiments indicate that workloads using TCUs have higher failure rates than workloads that avoid them. Other works have studied the impact of permanent faults in SAs in the context of DNN workloads. Still, these works did not consider the architectural features of TCUs or their internal structures [30–34].

Regarding number format characterization, several works in the literature have considered different emerging number representations (e.g., *posit*), which face the limitations of mature number formats, like *floating-point*, especially in new applications, such as those in the machine learning domain. However, most of these efforts are focused on the evaluation and comparison of the code format benefits, as well as the area and power costs of their implementations [35–40]. Some works have carried out experiments (in software) to address the implicit resiliency associated with the encoding representation of real numbers [41–43]. The experimental results suggest that posit is more error-resilient than FP. However, the impact of permanent hardware faults in the underlying hardware structures has not been analyzed and remains mostly unexplored.

In a previous work [44], we preliminarily analyzed the resiliency of TCUs affected by hardware defects in their internal structures (i.e., DPUs) and their impact on the execution

of GEMM workloads under two different number formats, *floating-point (FP)* and *posit*. Our preliminary results suggested that posit might be less sensitive to faults than standard representations (e.g., FP) by up to one order of magnitude. However, due to a TCU's intrinsic accumulative nature, the fault vulnerability and sensitivity of in-chip memory elements, format configurations (e.g., bit widths), and their impact on results were not fully explored in [44]. In the current work, we extend the reliability assessment of TCU accelerators in GPUs by considering the impact of faults affecting different structures inside the TCU cores (not only in the *DPUs'* logic but also in the *near registers'* storage elements), as well as their impacts on different bit-size configurations (16- and 32-bit) for two real number formats (i.e., *FP* and *posit*).

Our analysis was based on an open-source architectural hardware model of TCUs (*PyOpenTCU* [45]) that we developed, which supports several format configurations (e.g., 16- and 32-bit data). Moreover, the model includes an embedded fault injector tool to enable reliability evaluations.

In our experiments, we evaluated the complete execution cycle of TCUs when permanent faults affected the internal structures of TCUs (e.g., DPUs and near-registers). The evaluation required a total of 120 fault simulation campaigns, injecting around 200,000 faults per campaign. Overall, the complete fault simulation campaigns needed around 32 days. The experimental results indicate that hardware faults in TCUs accumulate their effects during the TCU's operative cycle, and in most cases, each fault corrupts 2 bits in the output results for both real number formats.

The major contributions of this work are as follows:

- We report the results of an extensive reliability assessment of permanent faults affecting the structures of an accelerator (TCUs) for AI workloads (e.g., GEMM) by considering two number format width configurations (i.e., 16- and 32-bit data) and two real number formats (i.e., *FP* and *posit*).
- We introduce *PyOpenTCU* [45], an open-source architectural model of TCUs in GPUs that integrates a custom reliability evaluation framework, which allows the evaluation of hardware defects located in the internal hardware structures of TCUs (i.e., DPUs and near-registers).
- We show that TCUs are highly sensitive to permanent hardware defects. However, we demonstrate that only around 5% to 10% of all analyzed and observed errors affect significantly the final result.
- We prove that the *posit* number format is less error-sensitive to permanent faults in comparison to the *floating-point* one by up to three orders of magnitude for 16 bits and up to twenty orders of magnitude in the 32-bit case.

Although we focused our analyses on and considered the architecture of TCUs in NVIDIA's GPUs, the proposed methodology can also be adapted and applied to the reliability assessment of similar machine learning accelerators inside GPUs, such as those in AMD or Intel GPU architectures (e.g., *matrix cores*).

The rest of this paper is organized as follows. Section 2 describes the related background regarding GPUs, TCUs, and floating-point data representations. Section 3 describes the evaluation methodology and introduces the study's framework. Section 4 presents the experimental results for every number format configuration. Finally, Section 5 concludes the paper.

2. Background

2.1. Organization of Graphics Processing Units (GPUs)

Modern GPUs are massive, general-purpose hardware accelerators able to provide high operative throughput by resorting to an array of homogeneous clusters of parallel cores (a.k.a. *streaming multiprocessors* or *SMs*). In modern GPU architectures, SMs are the primary execution unit, and they comprise up to four sub-cores to handle and boost the simultaneous execution of several threads (e.g., 32 threads or one *warp*) by resorting to a set of scalar processing cores and in-chip accelerators, such as *integer units* (*INT*),

floating-point units (*FPU*), *special function units* (*SFUs*), and *TCUs*. This variety of resources, in combination with register file banks, memories, and clever scheduling policies, allows the processing of large amounts of data with minimal latency effects. Typically, one SM comprises 32 to 64 INTs and FPUs, 4 SFUs, and 2 TCUs. In particular, the vector nature of TCUs involves the special and clever management of the running threads and memory resources in an SM, which is also extended to the GPU's ISA capabilities to operate it. The following subsection details and highlights the main operative features of TCUs as in-chip accelerators of GPUs.

2.2. TCUs' Organization and Operation in GPUs

Modern GPUs include in-chip accelerators to increase the performance during the execution of machine learning algorithms. In detailed terms, the most recent GPU architectures include accelerators for linear algebra focused on machine learning applications called *matrix core processing units* or TCUs [4,46]. Each TCU comprises one 4 × 4 array of *dot-product units* (DPUs), which are able to compute 16 *multiply-and-add* (MaA) operations per cycle on matrix segments (e.g., 4 × 4-size *A*, *B*, and *C* inputs) and perform M × M operations. The organization of the register file and the scheduling in GPUs with TCUs are adapted to provide a large number of operands to the TCUs and avoid latency effects. Thus, operands are organized as consecutive sets of registers during the execution of TCUs to exploit spatial locality. Moreover, GPUs include additional special registers (also known as buffers, *immediate registers*, *register file caches*, or *near-registers*) [47] as crucial elements to take advantage of temporal locality and improve the management of the matrix segments in the TCU's operation.

Since TCUs were designed to exploit the implicit parallelism inside GPU platforms, matrix tiling schemes and hardware scheduling policies have been implemented [48]. In particular, *thread grouping* schemes (a set of four consecutive threads per warp, or *thread groups*) cleverly and efficiently provide the data and support the management of a TCU's operation. In this scheme, some data operands (e.g., segments of Matrix C) are shared among the *thread groups* to efficiently provide operands to the TCU. In the GPU, these *thread groups* are executed in pairs, which represent an *octet*. In general, the octets and their associated data (matrix segments) are processed on TCUs, as is shown in Figure 1, where the octets are represented with the colors (*yellow*, *green*, *red*, and *violet*). Thanks to the octet data organization, these help hide the latency produced during the loading and storing process [4] by sharing the matrix segments among the thread groups per octet for their independent execution. In detail, each buffer is able to store the matrix segments (e.g., 4 × 4-size *A*, *B*, and *C* inputs) and the partial result of each M × M. Thus, the TCU operation and its accumulative process exploit the near-register architecture to reduce the time latency involved by reading and storing mechanisms between the SM registers and the TCU memory elements.

TCUs are flexible, and they support multi-precision and multi-size configurations. NVIDIA has introduced assembly instructions (e.g., *HMMA*) to process large matrix dimensions (e.g., 16 × 16) in TCUs as a sequence of operations. Each HMMA instruction has four operands, and each one uses a pair of registers. In this context, a sequence of instructions (i.e., *HMMA*) is used (see Figure 2) to allow the operation of 4 × 4 array segments. Then, consecutive instructions produce partial, accumulated, and final results. The partial and accumulated results are stored in the near-register file (buffers) for subsequent reuse with other input segments. Finally, the results are stored in the SM's register file.

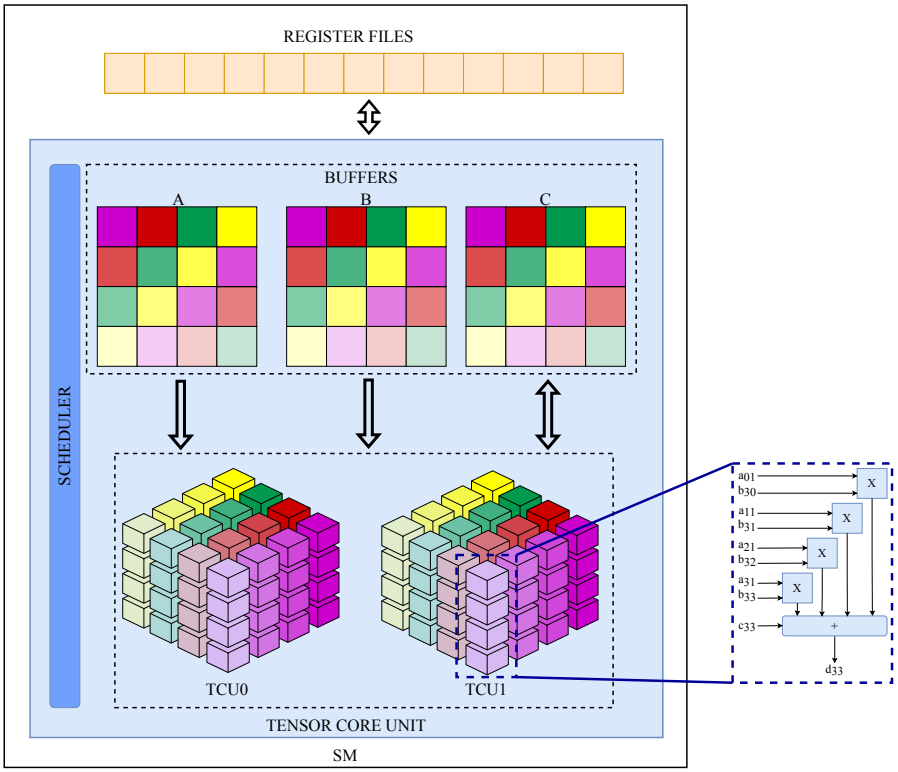

Figure 1. A general scheme of two TCUs inside an SM core. Each TCU is composed of a scheduler, 16 DPU cores, and its internal buffers (in this illustration, the matrix segments *A*, *B* and *C* and their octets are represented by the colors *yellow*, *green*, *red*, and *violet*. In detail, each field in the buffers stores four elements). Adapted from [4,46].

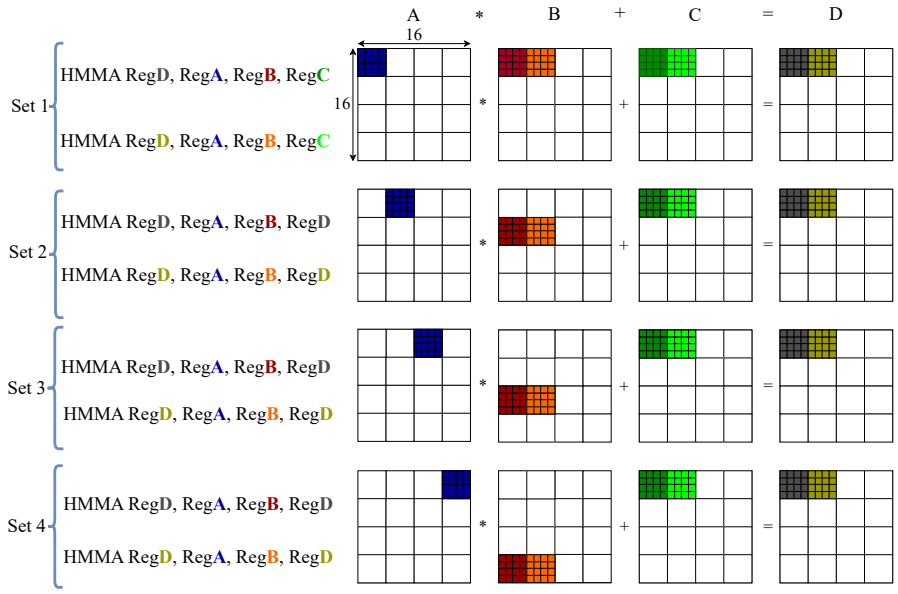

Figure 2. A general scheme of the M × M of the input segments (A, B and C matrix segments) and the instructions used by the *Thread Group* 0 to calculate two output segments in the 16 × 16 mode. The HMMA instructions are grouped in sets to process 4 × 4 input segments and handle the intermediate results. The resultant segment D is obtained after the sequence of four sets. The register banks (or buffers) serve as temporal accumulators for partial results (adapted from [4]).

2.3. Number Formats for Real Numbers

2.3.1. Floating-Point (FP) Format

This format encodes a real number into a binary representation code defined by the IEE-754 standard [49]. This standard establishes a codification using k bits (e.g., a 1-bit sign, w-bit exponent, and t-bit fraction). Mathematically, a real value can be described in the FP format using Equation (1), where *bias* is a constant offset used to normalize the exponent.

$$value = (-1)^{sign} * 2^{exp-bias} * (1 + \sum_{i=1}^{fn-1} b_{fn-1-i} * 2^{-i}) \quad (1)$$

2.3.2. Posit Format

This number format provides higher accuracy and a dynamic range, emerging as an alternative and drop-in replacement for the IEEE-754 format. The *posit* representation encodes any real number into a binary representation using N bits distributed in four fields (i.e., a 1-bit sign, r-bit regime, e-bit exponent, and f-bit fraction). This binary encoding mathematically describes any real number according to Equation (2).

$$value = (-1)^{sign} * useed^{k} * 2^{exp} * (1 + \sum_{i=1}^{fn-1} b_{fn-1-i} * 2^{-i}) \quad (2)$$

where $useed^k$ is a scale factor $useed = 2^{2^{\mathbf{es}}}$, and **es** is the exponent size. The parameter k is determined by the *run length* of 0 or 1 bits in the string of regime bits. The $useed^k$ value is used to compute the min_{pos} and max_{pos} values. Those values determine the posit dynamic range, maintaining it in perfect balance at about 1.0, and every *power of 2* has an exact reciprocal. Additionally, the posit format does not include redundancy representations or overflow/underflow cases during operations [50].

Some studies [51–54] have confirmed that, for some applications, like convolutional neural networks or CNNs, the posit format outperforms the FP one in terms of accuracy. Posit achieves superior accuracy in the range near 1.0, where most computations occur, making it very attractive for deep learning applications.

3. Evaluation of Real Number Representations in TCUs

This section introduces a method to evaluate and characterize the resiliency of TCUs and their crucial features, such as their number format, inside GPUs. Since our target was the assessment of the reliability features of TCUs and the possible effects of permanent faults (i.e., stuck-at-faults), we focused our evaluation on the architecture of TCU cores and the main propagation effects on individual scalar values. For this reason, our approach comprised three steps: a (i) functional characterization of TCUs, (ii) an evaluation of the impact of permanent faults in TCUs for two real number formats (i.e., *FP* and *posit*) under different format sizes, and (iii) an error impact assessment. Although the proposed method is described as referring to the NVIDIA TCUs, it can be exploited for the reliability analysis and evaluation of diverse machine learning accelerators, including AI architectures from AMD or Intel. Clearly, when changing the target architecture, the first step (*functional characterization*) must be adapted to match the key features of the micro-architecture under evaluation. The following subsections provide a detailed description of each step.

3.1. Functional Characterization of TCUs

In this step, we exploit the advantages of the functional/architectural simulation approach to evaluating AI workloads (e.g., GEMM) and their execution with TCUs. The evaluation of the effects of permanent faults affecting hardware accelerators is crucial for safety-critical applications, and this kind of analysis requires considerable computational power, considering the complexity of the structures to be studied and the long execution time of the application. Thus, the most feasible option to reduce the simulation time is to resort to architectural models.

We developed PyOpenTCU (https://github.com/TheColombianTeam/pyOpenTCU.git (accessed on 28 January 2024)), an open-source architectural simulator that implements the structural logic and the data organization of TCUs operating inside a GPU (including DPUs, memory elements, and controllers). This model provides the flexibility required to study the impact of hardware faults while still demanding feasible evaluation times, and it represents an effective solution to analyze the faults that may be affecting the different unit structures. The *PyOpenTCU* model is based on the architecture and descriptions from [4,46]. Furthermore, *PyOpenTCU* arises as a suitable tool for supporting the architectural design exploration of fault-tolerant structures in TCUs (e.g., hardware, software, or a clever combination of both). In addition, thanks to its programming flexibility, *PyOpenTCU* can be used to support the development of accurate error models and further evaluation in safety-critical application scenarios (e.g., autonomous robotics or automotive workloads through the accurate assessment of CNNs deployed on faulty TCUs).

Since TCUs are mainly composed of DPU cores and buffers (or near-registers), the TCUs in *PyOpenTCU* and their internal structures are described to perform the operations of each thread group independently. In detail, *PyOpenTCU* implements modern GPU architectures (e.g., NVIDIA Volta), which are composed of two TCUs per SM. Moreover, *PyOpenTCU* exploits the advantages of a high-level programming language (*Python*) to allow the exploration of different number formats, format sizes, and internal organization features, such as the dimension of the spatial arrays of DPUs. Thus, the architectural model supports the design exploration of different TCU solutions (e.g., changing the number format configuration). In detail, *PyOpenTCU* supports two number formats for real values (*FP* and *posit* formats) with two different configurations (16 and 32 bits). Those configurations are supported by *PyOpenTCU* and implemented using the *SoftFloat* and *SoftPosit* (https://pypi.org/project/sfpy/ (accessed on 28 January 2024)) libraries.

Unfortunately, a permanent fault arising in one of the TCUs might corrupt some or all threads executed on the faulty TCU (since the accumulative nature of the TCUs can propagate the corruption effects), which can be critical for some operations. In particular, the hardware reuse in the TCU is of high interest to understand how the error propagates or gets masked during the execution of M $\times$ Ms. Similarly, since the near-register structures are used to store operands (e.g., the A and B matrix segments), shared operands (e.g., the C matrix segments), accumulated/intermediate results, and final M $\times$ M results, the comprehension of how hardware faults placed in the memory elements and their error impact affect the workload execution is crucial for a TCU's resiliency. Due to this, and based on the general architecture scheme of TCUs shown in Figure 1, we identified two critical structures for the resiliency evaluation of TCUs: (i) DPU cores and (ii) buffers or near-registers.

In detailed terms, there are three locations (*inputs (INs)*, *outputs (OUTs)*, and *product results (PRs)*) inside the DPU where faults can be injected. When a fault occurs at the *IN* level, this fault corresponds to a faulty interconnection in the data path in charge of loading the data to be processed from the registers. Faults in the *PR* level mimic a fault defect in the internal units of the multipliers and their error propagation. Finally, if a fault affects any component of the tree adders, it can be modeled at the *OUT* level. On the other hand, we observed that the buffers have two important locations for evaluating the TCU resiliency: the input and output ports. In the case of a fault affecting the input ports, it represents a defect in the interconnection path between the register files and the near-register inside the TCU. If a fault affects the buffer output port, it mimics an internal defect inside memory cells.

3.2. Fault Evaluation and Error Propagation

In our evaluation, we developed a fault injection (FI) framework built on top of *PyOpenTCU* to analyze the impact of permanent (stuck-at) faults on TCUs and propagate their effects at the final output result (*the matrix affected*). In fact, the computational complexity of this kind of framework is related to the total number of hardware defects for evaluation.

In this regard, its complexity is represented as $O(n)$, where n represents the total number of faults under study. This framework is divided into four main steps: performing the ideal (golden or fault-free) application, fault list generation, injecting the faults (based on the pin-level fault injection strategy [55]), and classifying the results (i.e., comparing each result obtained for the faulty model with the fault-free result).

The first step (***performing the ideal application***) computes the fault-free M × M employing the TCUs. Then, the input and output matrices are stored to be processed in the next steps. Subsequently, the ***fault list generator*** step produces the list of target faults used in the fault simulation (FSim) campaigns. Each target fault is composed of a fault descriptor with five configuration parameters (fault target—FT, thread group ID—TGID, position—Pos, Mask, and stuck-at—ST). The TGID and Pos parameters identify the structure inside the TCU to be affected by the fault (e.g., the buffer or DPU). The FT parameter is associated with the target TCU structure to inject a fault (e.g., in the case of buffers, it represents the input or output port identifier, while in DPUs' case, it represents the internal structure). The other parameters identify the target bit affected by the fault (Pos and Mask) and the type of stuck-at fault (ST) to apply (0 or 1).

During the ***fault injection*** step, the framework reads one fault descriptor and places a fault inside a TCU. The framework mimics the fault effect via saboteurs [56]. In detailed terms, this step introduces defects in the mentioned locations (i.e., buffers ports and the input, output, and partial results of DPUs). It allows for the study of the error propagation to evaluate the fault impacts on the M × M level. Afterward, *PyOpenTCU* executes the M × M. The output result is collected for later analysis. Then, a new fault descriptor is selected, and the fault injection procedure restarts. As our evaluation is focused on the assessment of typical AI applications (e.g., GEMM) deployed on TCUs, and the TCUs produce results and operate in an accumulative scheme, our ***classification*** step is performed only after the end of the accumulative process (i.e., corresponding to the final computed matrix). Thus, the framework determines and stores the differences between the faulty results and the reference outputs.

As our evaluation intends to determine the effect of permanent faults and their impact when TCUs are running typical AI workloads (i.e., GEMM), one of the most important concerns is related to the input tile (*input matrices*) processed by the TCUs. High-precision applications, like CNN domains, require computations with small magnitudes (e.g., CNNs are used to operate weights and intermediate feature maps in the neighborhoods of ± 1.0 [57]). Moreover, some studies [58] have demonstrated that computations composed mainly of zeros are crucial to guarantee the proper operation of CNN applications. Since this operation does not work properly, the CNN starts to experience bias that affects its performance. For this reason, we performed our assessment of TCUs using the typical data tiles in CNN domains, which have a distribution in the range of ± 1.0. In detailed terms, we employed three types of tiles: *random (R)* tiles with a random distribution of values, *zero (Z)* tiles with numbers close to zero, and *triangular (T)* tiles with a triangular distribution of values. This selection is also in concordance with other works in the field that have argued that fault propagation in the GPU's data path is data-independent if the input data (tiles) are not biased (i.e., composed of an excessive amount of 0 s or all 1 s) [59].

It must be noted that *PyOpenTCU* implements an architectural description of TCUs inside GPUs. Thus, our fault characterization is based on functional, structural, and pin-level fault approaches. This evaluation might include some limitations, such as less accuracy for some structures (e.g., internal adders or multipliers) in comparison to low-level (RT- or gate-level) micro-architectural models. However, several works [60,61] have demonstrated that strategies based on this kind of approach (pin-level fault injections and architectural/functional simulations) offer an appropriate balance between accuracy and computation effort. Moreover, in our work, we employed an exhaustive and extensive evaluation with the purpose of reducing possible inaccuracies. Similarly, our evaluation mainly focused on the analysis of permanent (stuck-at) faults on TCUs. However, we

instrumented the *PyOpenTCU* tool to support further analyses of other fault types, such as temporal faults (e.g., single-event upsets), which were out of the scope of the current work.

3.3. Error Impact Assessment

Thereafter, our error impact evaluation was based on the discrepancies between the fault-free and the corrupted M × M results. Since this analysis depended on the TCU's configuration (i.e., number), first, we classified the impact of each fault on each TCU result value according to the following categories:

- *Masked*: The fault does not produce any effect;
- *Silent data corruption* (SDC): The fault affects the results by corrupting one or more output values;
- *Detected unrecoverable error* (DUE): The fault prevents the correct execution of the application (i.e., the results show one or more values of *inf* or *NaN*).

Afterward, our assessment performed a quantitative evaluation, which enabled us to compare each value and determine the magnitude of impacts produced by each permanent fault. Since our evaluation was intended to estimate the effect of hardware faults at the application level in AI workloads (i.e., GEMM), our approach resorted to the *relative error* (also known as the *relative uncertainty approximation error*) as an evaluation metric, which is the ratio between the value produced by a unit affected by a fault and its corresponding fault-free value. This metric allows us to determine the impact of a hardware fault at the application level (i.e., *the final output tile matrix*) since it can compare values of different magnitudes. Moreover, this metric provides a reliable evaluation, especially in terms of outlier cases, since all the employed values are normalized during their computation.

4. Experimental Results

We used the typical configuration of an SM with two TCUs. Thus, in *PyOpenTCU*, all warps (and their internal *thread groups*) could address both TCUs. For the evaluation, we use the configuration of 16 × 16 input matrices (i.e., a complete M × M requires a sequence of 8 HMMA instructions) with two highly used bit-sizes in ML workloads for both arithmetic formats (e.g., 16 and 32 bits). Our FSim campaigns were composed of 120 simulations (60 simulations per considered number format, with 10 simulation campaigns per input tile type and bit-width). Each exhaustive simulation campaign included a total of 196,608 faults, which corresponded to the total number of possible stuck-at faults in the DPUs per TCU (i.e., 16 DPUs, each one composed of 9 INs, 4 PRs, and 1 OUT), operating in both bit sizes (16 and 32) and the total of thread groups (8), and the possible faults affecting the buffers with byte-size of each one (3 kb). This evaluation demanded a computational complexity of 32.1 days. All experiments were performed on an Hewlett-Packard Z2 G5 workstation with an Intel Core i9-10800 CPU with 20 cores and 32 GB of RAM, which is available at Department of Control and Computer Engineering (DAUIN) from the Politecnico di Torino.

Our evaluation was divided into two main stages: (i) the understanding and classification of error effects from hardware faults arising on the structures of the TCUs and (ii) a quantitative evaluation of the errors caused by the fault effects on the results. The subsequent subsections provide a detailed overview of these stages.

4.1. Fault Effect Assessment

At this stage, we studied the error effects of a fault and its relation with each number format. In particular, we compared each element of a fault-free matrix with the obtained matrix after the simulation (*affected matrix*). Figure 3 depicts the distribution of fault effects for the DPUs and buffers in the TCUs. In detailed terms, we classified the impact of faults arising at different injection places (*In*, *RP*, and *Out*) for both formats (*posit* and *FP*) under two bit-sizes configurations, 16 bits *(the top)* and 32 bits *(the bottom)*. Our results indicate that the TCUs are highly sensitive to permanent faults. In fact, the results show that both number formats (FP and posit) exhibit a similar trend, and they also reveal that the bit width has a strong relationship with error propagation. Interestingly, our evaluation shows

that number formats with less bit width are more prone to propagating the effect of a permanent fault.

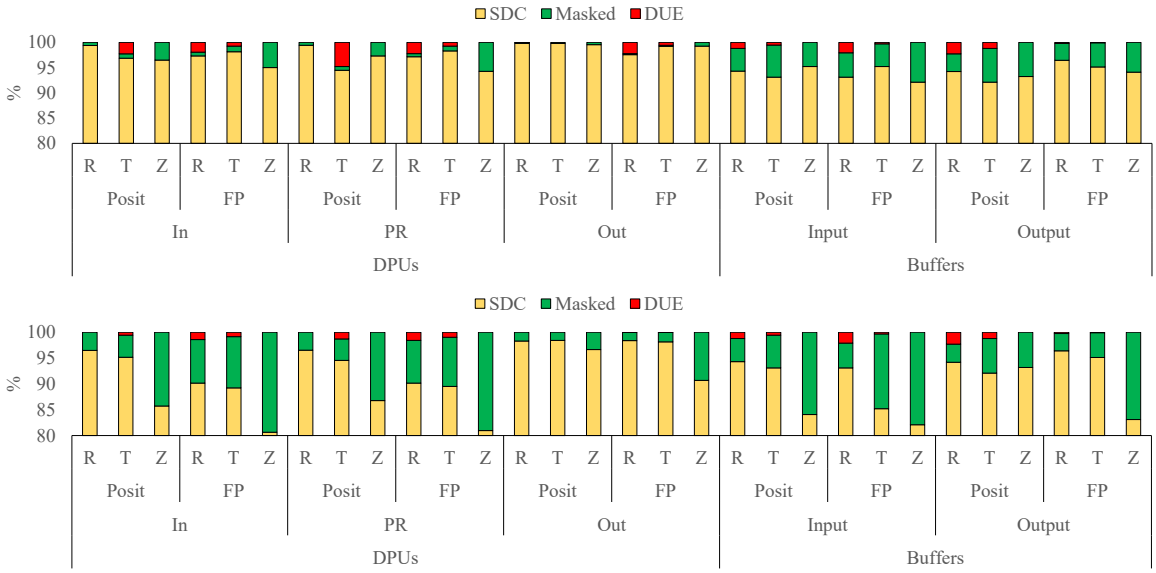

Figure 3. Distribution of fault effects alongside their impacts on the output result for all analyzed benchmarks for the TCU and buffer configurations of 16 bits (**the top**) and the TCU and buffer configurations of 32 bits (**the bottom**).

The observed behavior seemed to be associated with the length of the mantissa (FP) or fraction (posit) fields in the cores and the accumulative process during the execution of the TCUs. We observed that hardware faults affecting the corrupted TCUs mainly produced small errors observable on the mantissa and fraction fields. However, a deep analysis seemed to indicate that the accumulation process in the TCUs contributed to masking a considerable portion of the effects, especially in large bit-with formats (i.e., 32 bits), so the error propagation on those fields was mostly masked during the M $\times$ M operation.

In detailed terms, the M $\times$ Ms with 16-bit size operands suffered 97% of the faults causing SDCs and only 1% of the faults producing DUEs (e.g., not-a-number or NaN cases). Meanwhile, if M $\times$ M was performed on 32 bits, the SDCs corresponded to 92% of the faults injected, around 8% of the faults were classified as masked, and fewer than 1% were labeled as DUEs.

The results also show that, if the TCU is configured to perform M $\times$ M of 16-bit size, and faults arise in the DPUs' outputs (OUT), these faults mainly cause SDCs (up to 99%). Meanwhile, if a fault arises on the IN or PR places of the DPUs, the fault propagation probability and its associated impact decrease up to 96%. However, DUE cases might increase. In addition, faults affecting the buffer input port were prone to corrupting and propagating effects on the output matrix in up to 95% of the evaluated cases, whereas faults arising in memory cells of the near-register/buffer structures increased the fault corruption of the results by up to 97%. On the other hand, if the M $\times$ M were computed with a 32-bit size, and a defect affected the DPU's output (OUT), there would be up to a 98% probability of changing the expected value. However, the probability of error propagation decreases by up to 80% when a fault is placed on the IN. Therefore, defects affecting the near-registers (buffers) might change the final result by around 88% and 92% when the fault is placed in the input and output ports, respectively.

However, the results demonstrate that, when TCUs are configured to operate in a 16-bit configuration, the percentage of SDCs affecting the DPU's structures increases by up to 1.5% with respect to the percentage of faults affecting the buffers. In contrast, the buffer ports are up to 1% more sensitive than the DPUs' structures when M $\times$ M operations are performed on 32 bits.

Remarkably, the DUE probability is related to the type of processed input tile. The results show that no faults crash the application when the TCUs compute values close to zero, regardless of the number format configuration. In in-depth terms, we found that the DUEs mainly appeared when the fault affected the most significant bits. This means that the exponent (FP) and the regime (posit) are more prone to crashing the application. On the other hand, the masked cases were produced when the faults affected the less significant bits.

4.2. Quantitative Error Evaluation

Let us move on to the quantitative evaluation. Our approach estimated the impact of hardware faults when a fine-grain AI workload (e.g., GEMM) was considered by comparing the differences between the fault-free matrix and the observed one. For this reason, we employed the *relative error*, which allowed us to quantify the magnitude of error from the structural defects on the TCUs.

As expected, the experimental results proved that the fault impact was related to the most significant bits affected in the final result; see Figure 4. A detailed view shows that both bit widths (16 and 32 bits) exhibited a similar trend, regardless of the number format and the faulty TCU structure (i.e., DPU's internal structures and buffer ports). Thus, the exponent and the regime were more sensitive to permanent faults. Based on our results, the worst fault effect for 32-bit operations reached up to 10^{38} and 10^{18} in FP and posit, respectively. Meanwhile, M $\times$ M performed in 16 bits presented the highest error, up to 10^{8} in FP and 10^{5} for posit.

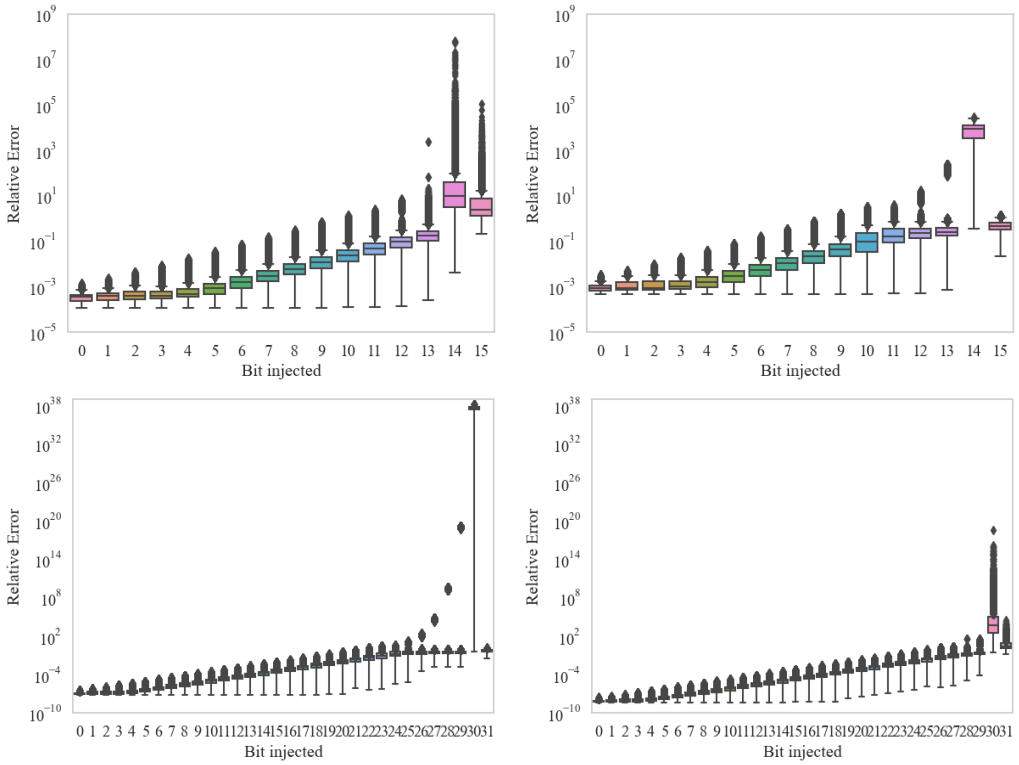

Figure 4. Relative error per affected bit. **The top** shows the impact of permanent faults when TCUs were operating on 16 bits, while **the bottom** shows the error magnitude if the bit width was 32 bits, for both number formats (FP on the **left** and posit on the **right**).

Nevertheless, for both bit widths, the changes affecting the exponent (in FP) and the regime (in posit) had a probability of less than 10%. Despite this, the fraction (FP) and the fraction and exponent (posit) were more likely to change one or more bits, and the error generated by these changes was less than 1.0. Indeed, we found that 90% and 95% of error

impacts were below 1.0 for the 16- and 32-bit widths, respectively (see Figure 5). In the CNN domains, errors of this size might be controlled or neglected [28,58,62].

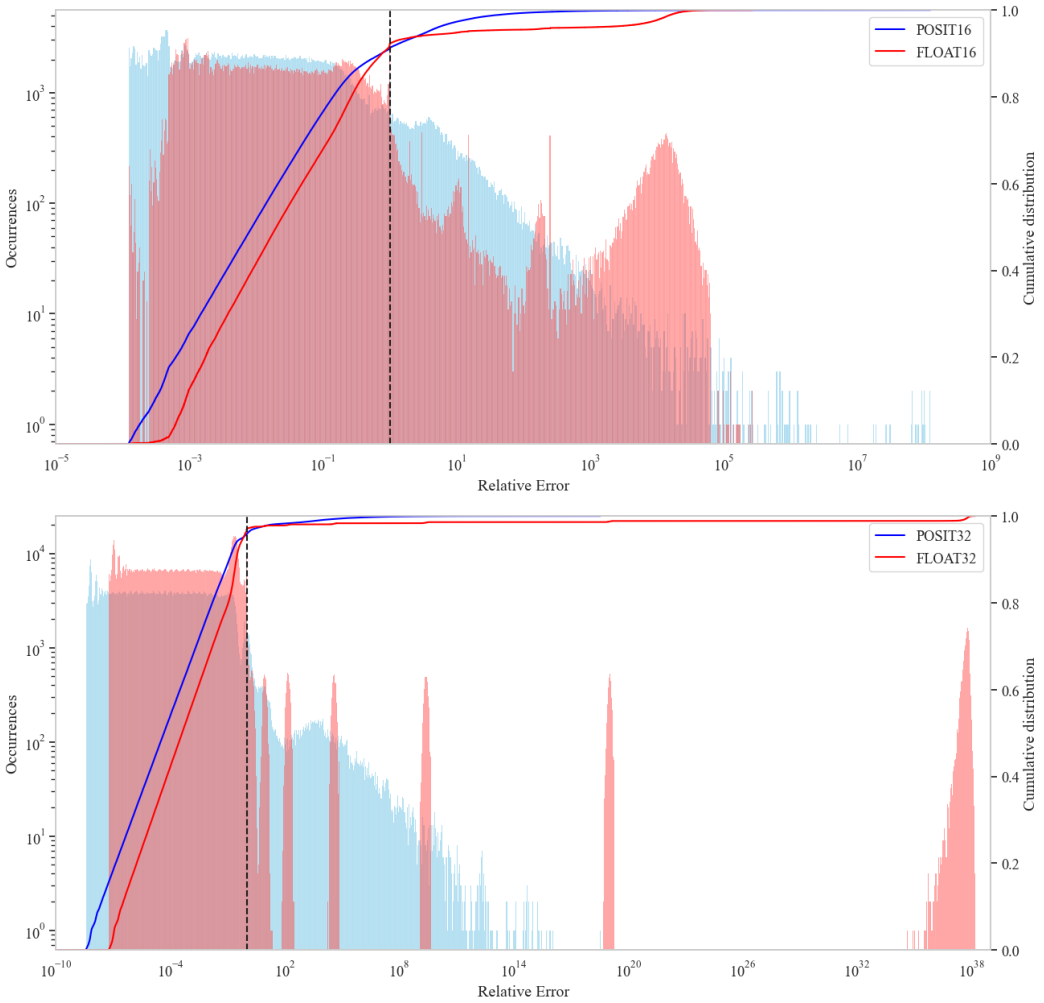

Figure 5. Relative error and cumulative distribution for both number formats (*posit* format is presented in blue, meanwhile *FP* on is shown in red). We reported the impact distribution when the output buffer ports were affected by faults and the TCUs were operating on 16 bits (**top**) and 32 bits (**bottom**). The black line represents an error magnitude of 1.0.

In detailed terms, the cumulative distribution of the errors shows that the fault impact was directly linked to the number format. A closer look confirmed that small error magnitudes were mainly produced by faults in the mantissa (FP) and the fraction (posit) fields. Meanwhile, large impacts were commonly obtained when the most significant bits (exponent in FP and regime in posit) were affected by a fault. Surprisingly, FP had an error distribution mainly concentrated around 10^1, 10^2, 10^4, 10^{11}, 10^{18}, and 10^{38}. On the other hand, we found that in posit, the error impact was widely distributed between 10^0 and 10^{18}, mainly due to impacts on the regime bit-fields.

Let us conclude. Our analysis highlights the relation between the real number formats and the distribution and magnitude of errors within TCUs that are affected by permanent faults. Interestingly, the experimental results prove that posit formats in TCUs produce failures with a lower order of magnitude compared to FP ones. Also, the results demonstrate the sensitivity to faults of exponent (FP) and regime (posit) fields. Our evaluation can be fruitfully exploited to support the design of reliable hardware accelerators for machine learning domains, taking into consideration design parameters as the number format and the bit width. Although the current work focused on the fault characterization of

hardware defects and their effect on the fine-grain operation of TCUs, our findings might be extended to support the reliability assessment of larger workloads using TCUs inside GPUs, such as dense machine learning in automotive and robotics applications. In this context, hybrid frameworks, which exploit the advantages of instruction-accurate simulators, like *PyOpenTCU*, and the fast and efficient functional execution of applications, such as CNNs, can provide a new opportunity to evaluate such computationally demanding and data-intensive applications.

5. Conclusions

This work has analyzed TCUs' resiliency inside GPUs. We developed a flexible architectural model named *PyOpenTCU* that allows the analysis and evaluation of the architectural impacts of permanent faults affecting TCUs. Resorting to this model, we performed an accurate reliability analysis with different TCU versions, based on extensive fault injection campaigns.

The experimental results show that the exponent (in the FP format) and the regime (in the posit format) fields are the most sensitive to faults, regardless of the bit width. The results also indicate that the FP format is more sensitive to permanent faults than the posit format (by up to three orders of magnitude for 16-bit operations and up to twenty orders of magnitude in the 32-bit case) for the analyzed operative ranges. Moreover, our experimental results suggest that 90% to 95% of the analyzed faults in the TCU structures produced errors with magnitudes lower than 1.0.

Furthermore, the results show that corrupting the most significant bit produces the largest error effect. In particular, our experiments also show that the most frequent effect on the outputs comprises the corruption of 2 bits in the output value when a permanent fault is present inside a TCU.

Additionally, our results indicate that the bit width is strongly associated with error propagation. Indeed, faults affecting TCUs configured for 32-bit width operations of M $\times$ Ms are less prone to propagate errors by up to 5% compared to 16-bit configurations of the TCU.

In future works, we plan to extend our assessment to evaluate the largest matrix sizes, such as those involved in feature maps or activation layers in CNNs. Moreover, we plan to use the findings from this work to devise effective hardening techniques.

Author Contributions: Conceptualization: R.L.S., J.-D.G.-B. and J.E.R.C.; methodology: R.L.S., J.-D.G.-B., J.E.R.C. and M.S.R.; software/hardware: R.L.S. and J.E.R.C.; validation, R.L.S., J.-D.G.-B. and J.E.R.C.; formal analysis: R.L.S., J.-D.G.-B., J.E.R.C. and M.S.R.; writing—original draft preparation: R.L.S.; writing—review and editing: R.L.S., J.-D.G.-B., J.E.R.C. and M.S.R.; visualization: R.L.S., J.-D.G.-B., J.E.R.C. and M.S.R. All authors have read and agreed to the published version of the manuscript.

Funding: This project is funded by the Ministery of University and Research via the "National Recovery and Resilience Plan (PNRR) through the National Center for HPC, Big Data and Quantum Computing".

Data Availability Statement: Data is contained within the article.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Peccerillo, B.; Mannino, M.; Mondelli, A.; Bartolini, S. A survey on hardware accelerators: Taxonomy, trends, challenges, and perspectives. *J. Syst. Archit.* **2022**, *129*, 102561. [CrossRef]
2. Dally, B. Hardware for Deep Learning. In Proceedings of the 2023 IEEE Hot Chips 35 Symposium (HCS), IEEE Computer Society, Palo Alto, CA, USA, 27–29 August 2023; pp. 1–58.
3. Jouppi, N.P.; Young, C.; Patil, N.; Patterson, D.; Agrawal, G.; Bajwa, R.; Bates, S.; Bhatia, S.; Boden, N.; Borchers, A.; et al. In-Datacenter Performance Analysis of a Tensor Processing Unit. *SIGARCH Comput. Archit. News* **2017**, *45*, 1–12. [CrossRef]

4. Raihan, M.A.; Goli, N.; Aamodt, T.M. Modeling Deep Learning Accelerator Enabled GPUs. In Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), Madison, WI, USA, 24–26 March 2019; pp. 79–92.
5. Dally, W.J.; Keckler, S.W.; Kirk, D.B. Evolution of the Graphics Processing Unit (GPU). *IEEE Micro* **2021**, *41*, 42–51. [CrossRef]
6. Lee, W.K.; Seo, H.; Zhang, Z.; Hwang, S.O. TensorCrypto: High Throughput Acceleration of Lattice-Based Cryptography Using Tensor Core on GPU. *IEEE Access* **2022**, *10*, 20616–20632. [CrossRef]
7. Groth, S.; Teich, J.; Hannig, F. Efficient Application of Tensor Core Units for Convolving Images. In Proceedings of the 24th International Workshop on Software and Compilers for Embedded Systems, Eindhoven, The Netherlands, 1–2 November 2021.
8. Oakden, T.; Kavakli, M. Graphics Processing in Virtual Production. In Proceedings of the 2022 14th International Conference on Computer and Automation Engineering (ICCAE), Brisbane, Australia, 25–27 March 2022; pp. 61–64.
9. Gati, N.J.; Yang, L.T.; Feng, J.; Mo, Y.; Alazab, M. Differentially Private Tensor Train Deep Computation for Internet of Multimedia Things. *ACM Trans. Multimed. Comput. Commun. Appl.* **2020**, *16*, 1–20. [CrossRef]
10. Fu, C.; Yang, Z.; Liu, X.Y.; Yang, J.; Walid, A.; Yang, L.T. Secure Tensor Decomposition for Heterogeneous Multimedia Data in Cloud Computing. *IEEE Trans. Comput. Soc. Syst.* **2020**, *7*, 247–260. [CrossRef]
11. Wang, H.; Yang, W.; Hu, R.; Ouyang, R.; Li, K.; Li, K. A Novel Parallel Algorithm for Sparse Tensor Matrix Chain Multiplication via TCU-Acceleration. *IEEE Trans. Parallel Distrib. Syst.* **2023**, *34*, 2419–2432. [CrossRef]
12. Chen, H.; Ahmad, F.; Vorobyov, S.; Porikli, F. Tensor Decompositions in Wireless Communications and MIMO Radar. *IEEE J. Sel. Top. Signal Process.* **2021**, *15*, 438–453. [CrossRef]
13. Xu, H.; Jiang, G.; Yu, M.; Zhu, Z.; Bai, Y.; Song, Y.; Sun, H. Tensor Product and Tensor-Singular Value Decomposition Based Multi-Exposure Fusion of Images. *IEEE Trans. Multimed.* **2022**, *24*, 3738–3753. [CrossRef]
14. Cheng, M.; Jing, L.; Ng, M.K. A Weighted Tensor Factorization Method for Low-Rank Tensor Completion. In Proceedings of the 2019 IEEE Fifth International Conference on Multimedia Big Data (BigMM), Singapore, 11–13 September 2019; pp. 30–38. [CrossRef]
15. Sofuoglu, S.E.; Aviyente, S. Graph Regularized Tensor Train Decomposition. In Proceedings of the ICASSP 2020—2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Barcelona, Spain, 4–8 May 2020; pp. 3912–3916. [CrossRef]
16. Zeng, H.; Xue, J.; Luong, H.Q.; Philips, W. Multimodal Core Tensor Factorization and its Applications to Low-Rank Tensor Completion. *IEEE Trans. Multimed.* **2023**, *25*, 7010–7024. [CrossRef]
17. Chen, L.; Liu, Y.; Zhu, C. Robust Tensor Principal Component Analysis in All Modes. In Proceedings of the 2018 IEEE International Conference on Multimedia and Expo (ICME), San Diego, CA, USA, 23–27 July 2018; pp. 1–6. [CrossRef]
18. Chang, S.Y.; Wu, H.C.; Yan, K.; Chen, X.; Wu, Y. Novel Personalized Multimedia Recommendation Systems Using Tensor Singular-Value-Decomposition. In Proceedings of the 2023 IEEE International Symposium on Broadband Multimedia Systems and Broadcasting (BMSB), Beijing, China, 14–16 June 2023; pp. 1–7. [CrossRef]
19. Liu, Y.; Yan, Z.; Tan, J.; Li, Y. Multi-Purpose Oriented Single Nighttime Image Haze Removal Based on Unified Variational Retinex Model. *IEEE Trans. Circuits Syst. Video Technol.* **2023**, *33*, 1643–1657. [CrossRef]
20. Lee, A. Train Spotting: Startup Gets on Track with AI and NVIDIA Jetson to Ensure Safety, Cost Savings for Railways. 2022. Available online: https://resources.nvidia.com/en-us-jetson-success/rail-vision-startup-uses?lx=XRDs_y (accessed on 28 January 2024).
21. *ISO 26262*; Road Vehicles—Functional Safety [Norm]. Available online: https://www.iso.org/standard/68387.html (accessed on 28 January 2024).
22. Mariani, R. Driving toward a Safer Future: NVIDIA Achieves Safety Milestones with DRIVE Hyperion Autonomous Vehicle Platform. 2023. Available online: https://blogs.nvidia.com/blog/2023/04/20/nvidia-drive-safety-milestones/ (accessed on 28 January 2024).
23. IEEE. *The International Roadmap for Devices and Systems: 2022*; Institute of Electrical and Electronics Engineers (IEEE): Piscataway, NJ, USA, 2022.
24. Strojwas, A.J.; Doong, K.; Ciplickas, D. Yield and Reliability Challenges at 7 nm and Below. In Proceedings of the 2019 Electron Devices Technology and Manufacturing Conference (EDTM), Singapore, 12–15 March 2019; pp. 179–181.
25. Libano, F.; Rech, P.; Brunhaver, J. On the Reliability of Xilinx's Deep Processing Unit and Systolic Arrays for Matrix Multiplication. In Proceedings of the 2020 20th European Conference on Radiation and Its Effects on Components and Systems (RADECS), Virtual, 19–23 June 2020; pp. 1–5.
26. Oml, P.; Netti, A.; Peng, Y.; Baldovin, A.; Paulitsch, M.; Espinosa, G.; Parra, J.; Hinz, G.; Knoll, A. HPC Hardware Design Reliability Benchmarking With HDFIT. *IEEE Trans. Parallel Distrib. Syst.* **2023**, *34*, 995–1006.
27. Rech, R.L.; Rech, P. Reliability of Google's Tensor Processing Units for Embedded Applications. In Proceedings of the 2022 Design, Automation & Test in Europe Conference & Exhibition (DATE), Virtual, 14–23 March 2022; pp. 376–381.
28. He, Y.; Hutton, M.; Chan, S.; De Gruijl, R.; Govindaraju, R.; Patil, N.; Li, Y. Understanding and Mitigating Hardware Failures in Deep Learning Training Systems. In Proceedings of the 50th Annual International Symposium on Computer Architecture ISCA '23, New York, NY, USA, 17–21 June 2023. [CrossRef]
29. Basso, P.M.; dos Santos, F.F.; Rech, P. Impact of Tensor Cores and Mixed Precision on the Reliability of Matrix Multiplication in GPUs. *IEEE Trans. Nucl. Sci.* **2020**, *67*, 1560–1565. [CrossRef]

30. Kundu, S.; Basu, K.; Sadi, M.; Titirsha, T.; Song, S.; Das, A.; Guin, U. Special Session: Reliability Analysis for AI/ML Hardware. In Proceedings of the 2021 IEEE 39th VLSI Test Symposium (VTS), San Diego, CA, USA, 25–28 April 2021; pp. 1–10.
31. Ozen, E.; Orailoglu, A. Architecting Decentralization and Customizability in DNN Accelerators for Hardware Defect Adaptation. *IEEE Trans.-Comput.-Aided Des. Integr. Circuits Syst.* **2022**, *41*, 3934–3945. [CrossRef]
32. Chaudhuri, A.; Talukdar, J.; Chakrabarty, K. Special Session: Fault Criticality Assessment in AI Accelerators. In Proceedings of the 2022 IEEE 40th VLSI Test Symposium (VTS), San Diego, CA, USA, 25–27 April 2022; pp. 1–4.
33. Agarwal, U.K.; Chan, A.; Asgari, A.; Pattabiraman, K. Towards Reliability Assessment of Systolic Arrays against Stuck-at Faults. In Proceedings of the 2023 53rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks—Supplemental Volume (DSN-S), Porto, Portugal, 27–30 June 2023; pp. 230–236. [CrossRef]
34. Tan, J.; Wang, Q.; Yan, K.; Wei, X.; Fu, X. Saca-FI: A microarchitecture-level fault injection framework for reliability analysis of systolic array based CNN accelerator. *Future Gener. Comput. Syst.* **2023**, *147*, 251–264. [CrossRef]
35. Stoyanov, M.; Webster, C. *Quantifying the Impact of Single Bit Flips on Floating Point Arithmetic*; Technical Report; Oak Ridge National Laboratory, Department of Computer Science, North Carolina State University: Oak Ridge, TN, USA, 2013.
36. Fu, H.; Mencer, O.; Luk, W. Comparing floating-point and logarithmic number representations for reconfigurable acceleration. In Proceedings of the IEEE International Conference on Field Programmable Technology, Bangkok, Thailand, 13–15 December 2006; pp. 337–340.
37. Haselman, M.; Beauchamp, M.; Wood, A.; Hauck, S.; Underwood, K.; Hemmert, K.S. A comparison of floating point and logarithmic number systems for FPGAs. In Proceedings of the 13th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'05), Napa, CA, USA, 18–20 April 2005; pp. 181–190.
38. Chugh, M.; Parhami, B. Logarithmic arithmetic as an alternative to floating-point: A review. In Proceedings of the 2013 Asilomar Conference on Signals, Systems and Computers, Pacific Grove, CA, USA, 3–6 November 2013; pp. 1139–1143.
39. Barrois, B.; Sentieys, O. Customizing fixed-point and floating-point arithmetic—A case study in K-means clustering. In Proceedings of the IEEE International Workshop on Signal Processing Systems (SiPS), Lorient, France, 3–5 October 2017; pp. 1–6.
40. Gohil, V.; Walia, S.; Mekie, J.; Awasthi, M. Fixed-Posit: A Floating-Point Representation for Error-Resilient Applications. *IEEE Trans. Circuits Syst. II Express Briefs* **2021**, *68*, 3341–3345. [CrossRef]
41. Schlueter, B.; Calhoun, J.; Poulos, A. Evaluating the Resiliency of Posits for Scientific Computing. In Proceedings of the SC '23 Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis, Denver, CO, USA, 12–17 November 2023; pp. 477–487.
42. Fatemi Langroudi, S.H.; Pandit, T.; Kudithipudi, D. Deep Learning Inference on Embedded Devices: Fixed-Point vs Posit. In Proceedings of the 1st Workshop on Energy Efficient Machine Learning and Cognitive Computing for Embedded Applications (EMC2), Williamsburg, VA, USA, 25 March 2018; pp. 19–23.
43. Alouani, I.; Khalifa, A.B.; Merchant, F.; Leupers, R. An Investigation on Inherent Robustness of Posit Data Representation. In Proceedings of the 34th International Conference on VLSI Design and 20th International Conference on Embedded Systems (VLSID), Guwahati, India, 20–24 February 2021; pp. 276–281.
44. Sierra, R.L.; Guerrero-Balaguera, J.D.; Condia, J.E.R.; Reorda, M.S. Analyzing the Impact of Different Real Number Formats on the Structural Reliability of TCUs in GPUs. In Proceedings of the 2023 IFIP/IEEE 31st International Conference on Very Large Scale Integration (VLSI-SoC), Dubai, United Arab Emirates, 16–18 October 2023; pp. 1–6. [CrossRef]
45. Limas Sierra, R.; Guerrero-Balaguera, J.D.; Condia, J.E.R.; Sonza Reorda, M. PyOpenTCU. 2023. Available online: https://github.com/TheColombianTeam/PyOpenTCU.git (accessed on 12 December 2023).
46. Boswell, B.R.; Siu, M.Y.; Choquette, J.H.; Alben, J.M.; Oberman, S. Generalized Acceleration of Matrix Multiply Accumulate Operations. U.S. Patent 10,338,919, 2 July 2019.
47. Gebhart, M.; Johnson, D.R.; Tarjan, D.; Keckler, S.W.; Dally, W.J.; Lindholm, E.; Skadron, K. Energy-efficient mechanisms for managing thread context in throughput processors. In Proceedings of the 38th Annual International Symposium on Computer Architecture (ISCA), San Jose, CA, USA, 4–8 June 2011; pp. 235–246.
48. Huang, J.; Yu, C.D.; van de Geijn, R.A. Implementing Strassen's Algorithm with CUTLASS on NVIDIA Volta GPUs. *arXiv* **2018**, arXiv:1808.07984.
49. *IEEE Std 754-2019 (Revision of IEEE 754-2008)*; IEEE Standard for Floating-Point Arithmetic. IEEE: Piscataway, NJ, USA, 2019; pp. 1–84. [CrossRef]
50. Gustafson, J.L.; Yonemoto, I. Beating Floating Point at Its Own Game: Posit Arithmetic. *Supercomput. Front. Innov. Int. J.* **2017**, *4*, 71–86.
51. Lindstrom, P.; Lloyd, S.; Hittinger, J. Universal Coding of the Reals: Alternatives to IEEE Floating Point. In Proceedings of the Conference for Next Generation Arithmetic CoNGA '18, New York, NY, USA, 15–18 July 2018. [CrossRef]
52. Mallasén, D.; Barrio, A.A.D.; Prieto-Matias, M. Big-PERCIVAL: Exploring the Native Use of 64-Bit Posit Arithmetic in Scientific Computing. *arXiv* **2023**, arXiv:2305.06946.
53. Mishra, S.M.; Tiwari, A.; Shekhawat, H.S.; Guha, P.; Trivedi, G.; Jan, P.; Nemec, Z. Comparison of Floating-point Representations for the Efficient Implementation of Machine Learning Algorithms. In Proceedings of the 2022 32nd International Conference Radioelektronika (RADIOELEKTRONIKA), Kosice, Slovakia, 21–22 April 2022; pp. 1–6. [CrossRef]

54. Murillo, R.; Del Barrio, A.A.; Botella, G. Customized Posit Adders and Multipliers using the FloPoCo Core Generator. In Proceedings of the 2020 IEEE International Symposium on Circuits and Systems (ISCAS), Seville, Spain, 12–14 October 2020; pp. 1–5. [CrossRef]
55. Gil, P.; Blanc, S.; Serrano, J.J. Book Chap. Pin-Level Hardware Fault Injection Techniques. In *Fault Injection Techniques and Tools for Embedded Systems Reliability Evaluation*; Benso, A., Prinetto, P., Eds.; Springer Science & Business Media: Berlin, Germany, 2003; pp. 63–79, ISBN 978-0-306-48711-8.
56. Jenn, E.; Arlat, J.; Rimén, M.; Ohlsson, J.; Karlsson, J. Fault Injection into VHDL Models: The MEFISTO Tool. In *Proceedings of the Predictably Dependable Computing Systems, Austin, TX, USA, 15–17 July 1995*; Randell, B., Laprie, J.C., Kopetz, H., Littlewood, B., Eds.; Springer: Berlin/Heidelberg, Germany, 1995; pp. 329–346.
57. Blundell, C.; Cornebise, J.; Kavukcuoglu, K.; Wierstra, D. Weight Uncertainty in Neural Network. In Proceedings of the 32nd International Conference on Machine Learning, Lille, France, 7–9 July 2015; Bach, F., Blei, D., Eds.; JMLR W&CP: Atlanta, GA, USA, 2015; Volume 37, pp. 1613–1622.
58. Češka, M.; Matyáš, J.; Mrazek, V.; Vojnar, T. Designing Approximate Arithmetic Circuits with Combined Error Constraints. *arXiv* **2022**, arXiv:2206.13077.
59. Previlon, F.G.; Kalra, C.; Kaeli, D.R.; Rech, P. A Comprehensive Evaluation of the Effects of Input Data on the Resilience of GPU Applications. In Proceedings of the 2019 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), Noordwijk, The Netherlands, 2–4 October 2019.
60. Zhang, Z.; Wang, Z.; Gu, X.; Chakrabarty, K. Physical-Defect Modeling and Optimization for Fault-Insertion Test. *IEEE Trans. Very Large Scale Integr. VLSI Syst.* **2012**, *20*, 723–736. [CrossRef]
61. Su, F.; Liu, C.; Stratigopoulos, H.G. Testability and Dependability of AI Hardware: Survey, Trends, Challenges, and Perspectives. *IEEE Des. Test* **2023**, *40*, 8–58. [CrossRef]
62. Jiang, H.; Santiago, F.J.H.; Mo, H.; Liu, L.; Han, J. Approximate Arithmetic Circuits: A Survey, Characterization, and Recent Applications. *Proc. IEEE* **2020**, *108*, 2108–2135. [CrossRef]