Reduced Precision DWC: an Efficient Hardening Strategy for Mixed-Precision Architectures

Fernando F. dos Santos, Marcelo Brandalero, *Member, IEEE,* Michael B. Sullivan, *Member, IEEE,* Pedro M. Basso Michael Hübner, *Senior Member, IEEE,* Luigi Carro, *Member, IEEE,* Paolo Rech *Senior Member, IEEE*

***Abstract*—Duplication with Comparison (DWC) is an effective software-level solution to improve the reliability of computing devices. However, it introduces performance and energy consumption overheads that could be unsuitable for high-performance computing or real-time safety-critical applications.**

In this work, we present Reduced-Precision Duplication with Comparison (RP-DWC) as a means to lower the overhead of DWC by executing the redundant copy in reduced precision. RP-DWC is particularly suitable for modern mixed-precision architectures, such as NVIDIA GPUs, that feature dedicated functional units for computing with programmable accuracy. We discuss the benefits and challenges associated with RP-DWC and show that the intrinsic difference between the mixed-precision copies allows for detecting most, but not all, errors. However, as the undetected faults are the ones that fall into the difference between precisions, they are the ones that produce a much smaller impact on the application output and, thus, might be tolerated.

We investigate RP-DWC impact into fault detection, performance, and energy consumption on Volta GPUs. Through fault injection and beam experiment, using three microbenchmarks and four real applications, we show that RP-DWC achieves an excellent coverage (up to 86%) with minimal overheads (as low as 0.1% time and 24% energy consumption overhead)

***Index Terms*—fault tolerance, mixed-precision architectures, graphics processing units, duplication with comparison**

I. Introduction

Reliability has been listed as one of the top 10 challenges to reach exascale computing by the United States Department Of Energy (DOE) [1], and is one of the primary constraints for safety-critical applications [2]. The reduced transistors' dimensions, the pursuit of low-power consumption, and the integration of several resources in a single chip has made modern devices extraordinarily efficient and powerful but has also exacerbated the probability of failures, in particular due to radiation-induced faults [3].

Parallel architectures such as Graphics Processing Units (GPUs), that are required to accelerate High-Performance Computing (HPC) applications and to detect objects in real-time safety-critical domains such as in self-driven vehicles, have been proved particularly vulnerable to transient faults [4]. Duplication With Comparison (DWC) is particularly suitable to address the reliability threat in these highly parallel devices, as it can be easily implemented in software and it has been proved very effective, detecting more than 90% of transient faults [4]–[6]. However, on GPUs, a traditional DWC approach imposes an average execution time or energy consumption overhead of 2x-2.5x [4], which is unsuitable for HPC and real-time safety-critical applications that also have strict performance and energy consumption constraints. Even advanced strategies, such as optimized software-directed instruction replication, still introduce an average overhead of 1.39x [6]. Redundant multithreading reduces the overhead, but only under the assumptions that there is no communication between threads and that unused resources are available [5].

In this paper, we move a step forward in the performance efficiency of DWC by presenting **Reduced-Precision DWC (RP-DWC)**, an improvement over the traditional DWC approach which consists in executing the replica in a lower precision. RP-DWC builds on ideas from previous works that have proposed to approximate the algorithm or hardware for efficient fault tolerance [7]–[10]. However, unlike existing solutions, RP-DWC does not require costly algorithm nor hardware modifications. RP-DWC is easily implementable with different granularities (correctness check at each or various operations) by the compiler and takes advantage of the intrinsic higher efficiency of low precision operations execution in Commercial Off-the-Shelf (COTS) devices. RP-DWC's efficiency is amplified in modern mixed-precision architectures, such as NVIDIA GPUs that feature dedicated functional units for the different precisions (see Fig. 1). In these devices, when executing the original high-precision application, the redundant mixed-precision hardware typically remains idle, and thus available to be leveraged for the parallel execution of a duplicated instruction flow in lower precision.

As the two copies in RP-DWC naturally provide different results, some faults will be undetectable. To evaluate the detection capabilities and performance overhead of RP-DWC, we perform, on Volta GPUs, beam experiments and software fault injection on a set of microbenchmarks and four essential applications used in HPC domains. The RP-DWC implementation in GPUs has very promising performance and energy consumption overheads that range, respectively, from 35% and 111% in a fine-grained implementation (correctness check at each operation) to 0.01% and 24% in a coarse-grained implementation (correctness check every 1,000 operations). As our data demonstrates, RP-DWC can achieve an average error detection of 74% (86% in the best case) under the beam and detects on the average 70% (76% in the best case) of the injected single bit flips. While RP-DWC detects fewer errors than traditional DWC, we show that the undetected errors

Fernando F. dos Santos, Pedro Martins Basso, Luigi Carro are with the Universidade Federal do Rio Grande do Sul, Brazil.

Paolo Rech is with Politecnico di Torino, Italy.

Marcelo Brandalero and Michael Hübner are with the Brandenburg University of Technology Cottbus-Senftenberg (B-TU), Germany.

Michael Sullivan is with NVIDIA, USA.

Manuscript received August 23, 2020

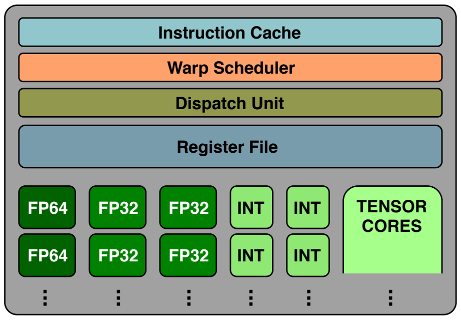


Fig. 1. The Volta GPU architecture features dedicated FP64 and FP32 cores.

fall inside the precision difference between the two copies, and, as our experiments show, they introduce at most a 0.1% variance in the application output value. These variances can be tolerated in many HPC applications and, as we show, are highly unlikely to induce failures in safety critical systems, such as miss-detections in object detection frameworks.

After an overview of the radiation effects on computing devices, some key concepts of mixed precision architectures, and an overview of related work discussed in Section II, **our work presents the following contributions:**

- We propose RP-DWC, a *software* fault detection strategy in which the redundant copy is executed in reduced precision, and an implementation of RP-DWC that effectively exploits the available low-precision GPU hardware that would otherwise waste energy without any contribution to algorithm execution (Section III).
- We investigate the limits of mixed-precision fault detection using RP-DWC and present a strategy to select a fault detection interval that enables good fault coverage without triggering false positives (Section IV).
- With the methodology described in Section V, we conduct comprehensive fault injection campaigns and neutron beam experiments using Volta GPUs
- We discuss the impact in HPC and safety-critical applications of the faults RP-DWC cannot detect and how RP-DWC would perform in devices without dedicated mixed-precision hardware.

II. Background and Related Work

In this section, we review some concepts and related work on the reliability of computing devices, approximate computing, and mixed-precision architectures useful to design the proposed RP-DWC technique.

A. Radiation Effects on Computing Devices

Ionizing particles such as atmospheric neutrons, protons, and heavy ions, may perturb a transistor's state, generating bit-flips in memory or current spikes in logic circuits that can potentially impact the functionality of computing devices. Neutron-induced events typically are *soft* or *transient faults*. They can induce the following effects on the program output:

- no effect: the fault is *masked*;
- an incorrect program output: the event is labeled a *Silent Data Corruption* (SDC);
- a change in the program behavior that is beyond merely the outputs, such as a crash or a device reboot: the event is labeled a *Detected Unrecoverable Error (DUE)*.

SDCs are particularly critical, as their silent nature makes them extremely hard to detect and they can potentially lead to unstable or unknown system states. Previous studies have already evaluated the reliability of CPUs, ARM, heterogeneous devices, FPGAs, GPUs, and Xeon Phi through radiation experiments [4], [11], fault-injection [12], [13], or both [14]–[16].

Memory arrays can be efficiently protected with Error Correcting Codes (ECC), which have already been shown to improve the device reliability significantly [3]. When it comes to computation, one of the most effective solutions to detect and mitigate transient faults is duplication or, in general, replication in software or hardware. By comparing the output of two independent copies of a program, it is possible to detect most of the transient faults. We discuss and compare such approaches in detail in Section II-C.

B. Approximate Computing and Mixed-Precision

Executing an instruction in double-precision floating-point (FP64) can take 2x longer and consume 2x the energy compared to the execution in FP32 [19]. Approximate computing exploits the applications' inherent error-resilience to improve the performance and energy efficiency by reducing the precision of data and operations [20]. The error resilience arises from the absence of a perfect result (such as in heuristic complex optimization problems) or from the application's ability to self-heal despite computational errors (e.g., simulations that end when the difference between iteration is small) [21].

Recently, to improve the efficiency of computation, approximate computing has been applied to both HPC and safety-critical domains, including object detection frameworks for autonomous vehicles [22]. Good object detection accuracy can be achieved by representing data in half-precision floating-point (FP16) or even in short integer format (8 bits) [23].

The adoption of approximate computing in a growing number of applications has pushed the demand for mixed-precision platforms. Most modern computing architectures such as FPGAs, GPUs, and CPUs are mixed-precision, supporting at least two of the five binary floating-point formats defined by the IEEE 754 (FP16, FP32, FP64, FP128, and FP256) [24]. In addition, many recent architectures support extensions featuring non-IEEE-compliant reduced-precision floating-point formats. Brain Float 16 (BF16) is a format with an 8bit exponent and 7bit mantissa; it is supported by extensions for the Power10 CPU, the NVIDIA A100 GPU, recent x86 CPUs (e.g. Intel Cooper Lake), and many DL accelerators. Tensor Float 32 (TF32) is a format with an 8bit exponent and 10bit mantissa; it is supported in the NVIDIA A100 GPU [25].

In a mixed-precision architecture, the user can select the precision of operations and data, tuning the hardware utilization and the execution time with the applications' needs. In some architectures, such as Intel x86, different precision operations are executed in the same core. Even if the hardware for different precision is the same, lower precision operations can still be executed much faster than higher precision operations.

TABLE I
COMPARISON OF THE MOST RELEVANT RELATED WORK ON FAULT TOLERANCE.

Approach	Work	Generic	Additional Hardware	Overhead		Error Coverage	Evaluation	
				Time	Energy	† = *correction*	F.I.	Beam
DWC for GPUs	[4]	Yes	No	∼200%-250%	N/A	80%-95%	No	Yes
Redundant Multithreading	[5]	Yes	No[1]	∼10%-200%	∼10-200%	N/A	No	No
Instruction replication (SInRG)	[6]	Yes	No[1]	∼36%	N/A	∼87% (Instr.)	Yes	Yes
Algorithm approximation TMR	[10]	No	No	∼33%-309%	N/A	∼30%†	No	Yes
HW unit and checker approximation	[7]–[9], [17], [18][2]	Yes	Yes	N/A	∼50%-140%	∼20%-40%	Yes	No
RP-DWC		Yes	No	0.01%-35%	24%-111%	55%-76%	Yes	Yes

Other architectures, such as NVIDIA Pascal, Volta, or Turing GPUs, have dedicated hardware to execute operations in double, single, and half precision [19].

Recently, the reliability of mixed-precision architectures has been evaluated through fault injection and radiation beam experiments [11]. Preliminary results show that, for GPUs, the higher the precision, the higher the expected error rate. The observed reduced reliability is mainly caused by the much larger area of higher precision computing cores, which implies a higher probability of being hit by a particle.

C. Related Work and Contributions to the State-of-the-Art

The main characteristics of an error detection strategy, besides efficacy (i.e., error coverage), are the overheads it introduces, if it is generic or valid only for an algorithm, and if it requires additional hardware to be implemented. Table I lists and summarizes the characteristics of the most relevant strategies proposed, at different levels of abstractions, to detect transient faults. These are discussed in detail next.

DWC is generic and, as demonstrated in [4]–[6], in GPUs, it can detect more than 95% of SDCs. Unfortunately, DWC is far from being efficient as the introduced overhead in terms of power consumption, silicon area, or execution time can be extremely high (at least 200% the unhardened version). As a result, DWC is impractical for HPC and real-time systems such as autonomous vehicles.

Recent work has proposed solutions to significantly reduce the cost of DWC in GPUs, maintaining its generality. With redundant multithreading, for some codes a slowdown of less than 10% can be achieved, but, because of the cost of communication, the overhead for other codes is still higher than 200% [5]. Software-managed Instruction Replication for GPUs (SInRG), on the contrary, can reach an overhead that is, on the average, 36% of the unhardened code execution time, covering about 87% of dynamic instructions (no absolute error coverage is provided) [6]. The reduced overhead of efficient DWC approaches proposed to date is guaranteed only under the assumption that there are sufficient computing resources to run the duplicated copy in parallel with the original one. Such an assumption is always valid for RP-DWC, as it leverages the available (otherwise idle) mixed-precision cores to execute the replica (details in Section III).

[1]These works also propose hardware modifications that can further reduce the overhead. Since RP-DWC is a software-only technique, we choose to include only the software-only results for these works.

[2]This work only evaluates permanent fault coverage.

Previous works have attempted mixed precision solutions to reduce the energy consumption of DWC, at algorithm or hardware level, to fault tolerance. The work in [10] proposes an approximate software-level Triple Modular Redundancy (TMR) for successive approximation algorithms, using loop-perforation to reduce the execution time of the replicated software. The higher the approximation, the lower the overhead, which can be as low as 33%. Unfortunately, the approach is limited to just a single class of algorithms and is not generic.

At the hardware level, other approaches have exploited adding approximate circuits to reduce the duplication overhead [7]–[9], [26], [27]. Work in [7] proposes to design a dedicated approximated hardware circuit for detecting errors in specific operations, while in [8], [9] authors design approximated checkers to catch just errors affecting the most significant portions of the mantissa or just the exponent. Work in [28] proposes duplicating floating-point units using reduced precision hardware to check for *permanent* errors (the proposed RP-DWC focuses on *transient* errors), and takes a mathematical approach to bound these errors. As additional and dedicated hardware is needed for error detection, the user has the flexibility of selecting the precision of the redundant copy considering the trade-off between fault coverage and overhead. Unfortunately, the additional hardware requirements increase the implementation cost of the strategy significantly. The work reports an area overhead of 30% - 90% and energy overhead of 50% - 140% for the lowest and highest precision copy, respectively. The reported *permanent* faults coverage ranges between 20% and 40%. The advantage of the approach lies in the ability to detect errors in the most significant bits. These works serve as a solid background for the application of approximation for error detection, in which we apply for *transient* error detection using COTS hardware.

Compared to all the fault-detection approaches summarized in Table I, **RP-DWC is the only approach that is, at the same time, generic**, **software-implemented** (requiring no changes to the hardware), and that **leverages the existing redundant mixed-precision hardware** for reduced performance and energy overheads. As such, it requires no prior knowledge of the algorithm, no hardware modifications, and can be automatically inserted by the compilation toolchain. *Compared to previous software-implemented instruction replication* [5], [6], we exploit the dedicated mixed-precision functional units available in modern GPUs to execute the replicated dataflow, leveraging these units that would otherwise be idle to increase the parallelism and to reduce the execution time of the hardened software. *Compared to previous approaches* *that propose approximate hardware* for fault detection, our approach is implemented entirely in software and targeted towards already existing and future architectures, which will feature mixed-precision components. A key advantage of RP-DWC is that the extra hardware (already paid in the form of mixed-precision processing elements) can be used both for performance improvements (when reliability is not an issue) as well as for approximate fault detection. Moreover, software implementations offer additional flexibility, allowing it to be implemented for both integer (INT64, INT32, INT16, INT8) as well as floating-point computations (FP64, FP32, FP16 and the new BF16 and TF32) being parameterized for different trade-offs between error detection and overheads.

Finally, as we will show in Section VI, our evaluation of the efficacy of approximation in detecting errors goes a step beyond previous works on approximate replica strategies by considering both beam experiments and fault injection, showing also the impact of detected and undetectable transient errors in the output correctness.

III. Reduced Precision DWC

In this section, we first give an overview of the main concepts of RP-DWC and then focus on its implementation on current and upcoming mixed-precision GPUs. We highlight the possible benefits RP-DWC brings, discuss its intrinsic limitations, and implementation challenges.

A. Main Concepts

RP-DWC consists of duplicating the instruction flow for execution in a lower precision. Using a reduced precision replica to implement DWC has three main benefits compared to a traditional DWC: (1) **Smaller overhead**. For example, an FP32 operation imposes about half the overheads of the equivalent FP64 operation. In the specific case of modern GPUs addressed here, since spare mixed-precision units are available, these overheads can be reduced even further. When no dependence between the two replicas exists (intrinsically true for any DWC-based strategy), and the code is not shared-memory or register limited (i.e., if the original code does not use all of the available registers or memory), then two replicas can be executed in parallel. (2) **Lower probability of having the replica corrupted**, as reducing precision has the effect of reducing the code error rate [11]. We can expect half the chance to have detections caused by errors in the replica when using RP-DWC with respect to traditional DWC. (3) **Diversity of the copies**, since the replicated operations will now execute in a different precision and use different processing resources, reducing the chances for a fault to have the same impact in both copies and remain undetected.

We proceed with a few mathematical definitions to formalize the RP-DWC approach. For simplicity and to ease the understanding of the implementation on modern GPUs we consider, without loss of generality, an FP64 original code and an FP32 redundant copy. The same definitions and considerations apply to other precisions. Let us consider two real numbers x and y, which can be encoded either as FP64 (x_{64} and y_{64}) or in FP32 (x_{32} and y_{32}). Operating x

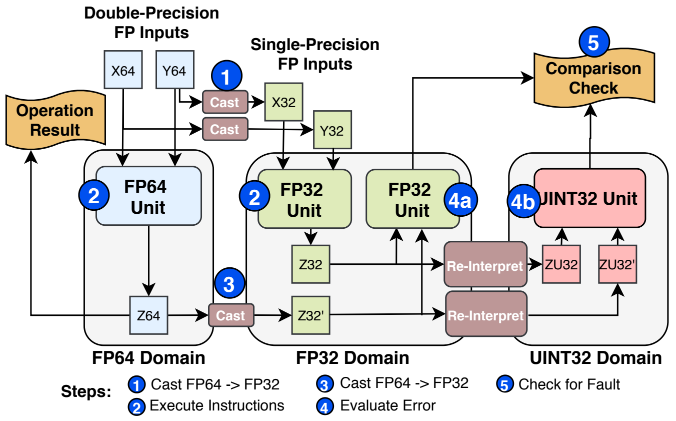


Fig. 2. Overview of the implementation.

and y together yields value z_{64} (the output of the original application) or z_{32} (used only for fault-detection). Even in the absence of faults, due to the reduced precision, $z_{64} \neq z_{32}$. RP-DWC defines a tolerable interval for the difference between z_{64} and z_{32}, named *Expected Precision Loss* (EPL), over which error detection is triggered. EPL acts as a threshold for triggering fault detection, and finding its value is among the challenges we address in Section IV. The intrinsic difference between z_{64} and z_{32} does not allow for the detection of all the faults. However, as we will show in the results, most of the faults are still detected with RP-DWC, and the undetectable faults might be tolerated as they produce an error inside the precision difference between FP64 and FP32 operations.

B. Overview of the Implementation

RP-DWC is a software technique to be applied at compiler time. To implement RP-DWC, the following procedures, shown in Figure 2, must be applied to each existing full precision instruction from the original program flow: cast the input values to reduced precision, execute the replicated instructions, cast the high-precision output to low-precision, and compare the mixed-precision results. We target modern GPU architectures to show RP-DWC implementation and how we leverage the mixed-precision hardware to further amortize DWC costs. Section VI-E discusses how RP-DWC would perform in the absence of dedicated mixed-precision cores.

Step 1. Casting the inputs to Reduced Precision. The 64 bits input data (x_{64} and y_{64}) needs to be reduced to 32 bits (x_{32} and y_{32}) to feed the reduced-precision replica. There are various ways to perform the cast operation. For instance, NVIDIA features four different casts (round up, round zero, round to nearest, round towards zero) [19]. The cast operation does not significantly impact the Expected Precision Loss, as in the experiment we performed the differences between the casts were, on average, orders of magnitude smaller than 1%. We select the default *round to nearest* cast. The value is taken from the input registers of the FP64 copy, and the casted value is stored in registers that are serving as input of the FP32 copy. While RP-DWC does not increase the pressure on caches or main memory, it increases the pressure on the register file, possibly increasing the overhead if there are insufficient registers to hold the values for both the FP64 and FP32 copies. This property is intrinsic of any software DWC for GPUs, but, as the FP32 copy requires only half the registers compared to the FP64 copy (each FP64 register occupies two FP32 registers), RP-DWC is less likely to reach register saturation than a traditional DWC (details in Section VI). Additionally, as in any DWC, if the fault affects the data *before* replication, then both copies will receive an erroneous input, preventing detection. On GPUs that include ECC in the main memory structures we can assume that the probability of having such a corruption in the input data is very low.

Step 2. Executing the original and reduced-precision instructions. The two copies are executed, either sequentially or in parallel in case dedicated mixed-precision hardware is available. RP-DWC for GPUs duplicates the instructions inside a thread rather than duplicating threads, warps, blocks, or kernels. By keeping the duplicated instructions inside the same thread of the original copy, we ensure that those are executed in the same core using the idle mixed-precision hardware, avoiding dependencies, duplicated caches, and synchronization issues. Since it is only necessary to duplicate the instructions, then add a comparison at the end of the thread or after an instruction code block, the duplication process can be automated at the compiling stage.

Step 3. Casting the high-precision result to reduced precision. Once both instructions complete their execution, z_{64} is cast down to FP32 to prepare the comparison. We cast z_{64} to FP32 as we just need to detect an error, and we will not have a more accurate error detection by performing the comparison in FP64.

Step 4. Performing the error detection. Dissimilarly to a traditional DWC, as the two copies z_{64} and z_{32} are naturally different, in RP-DWC, a simple a bit-wise comparison does not suffice to detect faults. We need to decide if the difference between the two copies is within the Expected Precision Loss (EPL), which, for now, we consider as given (Section IV-A will present an approach to determine EPL). The two outputs can be compared using the relative difference $\delta_r = \frac{z_{32}}{z_{64}}$ (with z_{64} casted to FP32, in Step 4a). The operation performed in RP-DWC (a division) is much more complex than in a traditional DWC (a comparison). The absolute difference would reduce the error detection overhead, but the result would depend on the exponent (subtracting similar numbers with a high exponent provides a huge absolute difference).

In the specific case of NVIDIA GPUs, there are twice as many FP32 cores as FP64 cores (see Fig.1 [19]). As RP-DWC uses one FP32 core per each FP64 operation in the original copy, there are surely enough FP32 cores available to perform the division in parallel with the next FP64-FP32 execution (Step 4a in Figure 2). NVIDIA also features a fast division operation (*fdividef*) that approximates the division results and takes half the time of a normal division. The use of *fdividef* rather than a normal division to detect faults is suggested as the relative difference between the results is negligible.

An alternative comparison is the use of unsigned integers. To compare z_{64} (now cast to FP32) and z_{32}, we can consider their representation as an unsigned integer (UINT32) and subtract them, which produces the difference δ_{UINT} (see step 4b in the figure). Re-interpreting the FP32 values as UINT is not a cast operation since it requires no modification in the representation, and does not significantly increase the overhead. By subtracting the two 32 bits representations interpreted as UINT gives a fast (and accurate) evaluation of the magnitude of the difference between the two numbers, which is exactly what is needed to detect errors. The higher the result of the subtraction, the more significant is the difference between the two representations. If the two representations differ only for some bits in the least significant digit of the mantissa, for instance, their values as UINT are very close. On the contrary, if the two floating-point values have different exponent or sign bit, their values as UINT are very different. As for δ_r calculation, the UINT subtraction (Step 4b in Figure 2) can also be performed in parallel with the next FP64-FP32 as GPUs have dedicated integer cores (see Fig.1).

The use of δ_{UINT} and δ_r can be chosen by the user. From our experience, reported in Section VI, δ_{UINT} is faster, but might detect fewer errors than δ_r if the exponent is very low.

Step 5. Comparing and taking action on the result. Once z_{64} and z_{32} are compared, the difference between z_{64} and z_{32} can be either (1) within or (2) outside the EPL.

In case (1), we assume that the instruction execution is successful or, if a fault happened, it produced an undetectable error. The execution can then proceed, after casting the next FP64 input to FP32. It is necessary always to feed the FP32 copy with the FP64 values to reduce the divergence of reduced-precision executions. In case (2), an error flag is raised to inform the application that an error has been detected.

As in any other duplication strategy, the comparison operation could be a single point of failure, i.e., a fault in this operation can lead to undetectable errors. Nevertheless, as we experimentally show in Section VI, this does not undermine the error coverage of RP-DWC as the comparison operation is less likely to be corrupted than the duplicated operation. With the granularity discussed in Section III-C we reduce the probability of a fault in the comparison operation even further.

Depending on the system implementation, once the error flag is detected, the execution can be terminated or rolled-back to the previous check. The roll-backed execution should be performed in FP64 by both copies to guarantee, for instance, that the detected error does not come from a round-off error. The approach to setting EPL discussed in Section IV-A reduces the rate of false-positives for any input the code may receive. However, one can decide to restrict the threshold if inputs are very well structured (for example, in image processing applications, the pixels' colors are well limited, and in physical simulations, the possible values are known). If an unexpected input is received, a false-positive can trigger error detection and re-execution will not solve the problem. Additionally, roll-back also can deal with the unlikely catastrophic cancelations, as detailed in Section IV-B. While executing both copies in FP64 might seem to undermine the efficiency of RP-DWC, that only happens when an error (or an unlikely false-positive) is detected, and in these few cases, the overhead would be identical to that of a traditional DWC.

C. Granularity of the Approach

DWC can be implemented with different granularities, i.e., performing the correctness check at each instruction or after a block of instructions. In a coarse-grained RP-DWC a sequence of FP64 instructions is duplicated with a sequence of FP32 instructions. The replicated sequence receives the FP64 input cast to FP32. The two sequences are then executed in parallel without interacting until reaching the correctness check.

To detect errors as early as possible, the correctness should be checked at each instruction. However, this requires a cast and a comparison operation at each operation, increasing the introduced overhead (which, as shown in Section VI, is still lower than a traditional DWC) and the probability for the error to occur in the comparison itself.

To evaluate the trade-off between the introduced overhead and the achieved error detection, we have implemented the correctness check with various granularity: at each instruction, at the end of the computation, and after some or several instructions (10, 100, 1,000). We do not consider longer sequences as basic blocks in GPUs have, on the average, between 20 to 100 instructions [29]. It is worth noting that, independently on the chosen granularity, a correctness check should be performed before any data-driven condition statement. A small divergence between FP64 and FP32, eventually caused by intrinsic differences, in fact, could force the two copies to take different paths.

A longer block of instructions can potentially increase the intrinsic difference of the two copies and, thus, the EPL. A bigger EPL, in principle, implies a higher number of undetectable errors. However, as the error propagates in the sequence of instruction, it may increase in magnitude, thus becoming detectable even with a bigger EPL.

IV. Error Detection Limitation

In this section, we present the challenges and a heuristic for choosing the EPL interval (Section IV-A) and the possible error detection limitations of RP-DWC (Section IV-B).

A. Estimating the Expected Precision Loss

One of the challenges for RP-DWC is to estimate the Expected Precision Loss (EPL) between two operations carried out in different precisions in order to be able to distinguish between an error (due to a fault) and an intrinsic loss of precision (due to reduced precision computation). The *exact* EPL depends on the inputs and the operation; for instance, if the exact input to two mixed-precision FP64/FP32 ADD operation is known, then the *exact* EPL (which we shall also refer to as the *ideal* EPL) is *a number* obtained by running the fault-free FP64/FP32 operations and comparing the outputs. If the input or the operation changes, however, the EPL will also change. In other words, the *ideal* EPL is different for each instance of operation while running an application, and is impractical to be used in real applications, since doing so would require computing a fault-free version of each FP64/FP32 operation with each new input at run time.

A viable alternative is to use a *sub-optimal* EPL, which we define no longer as an *exact number* for each different input and operation, but as an interval over which the outputs may diverge (given a few assumptions about the inputs) for each different operation. Since RP-DWC uses the EPL as a comparison instrument to flag faulty executions (when the difference between z_{64} and z_{32} falls outside of the EPL interval), it is critical to achieve a reasonable estimation of this interval. If the EPL is set to an interval that is *too narrow*, then some intrinsic differences between mixed-precision computations will be detected as errors, and a high rate of *false positives* will emerge. If, on the other hand, the EPL is set to an interval that is *too wide*, then a few errors may remain undetected, and a high rate of *false negatives* will emerge.

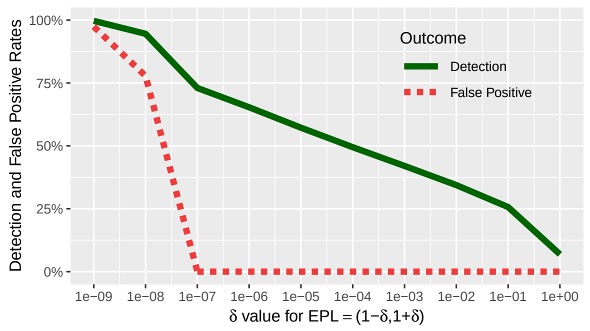


Fig. 3. Effect of the EPL interval onto the fault detection for ADD and MUL (the two curves overlap).

To exemplify this discussion, we execute 10M independent FP64 ADD operations and 10M independent MUL operations with random inputs in the range $(0, 1)$. Each one of the 10M ADD or MUL has its own *ideal* EPL, all being in the order of 1×10^{-7}. We then define *sub-optimal* EPL intervals in the form $I = (1 - \delta, 1 + \delta)$ for the relative error between the same operation executed in FP64 and FP32. We then injected faults and evaluated how well do each of those *sub-optimal* EPL intervals work for fault detection. The results, presented in Fig 3, show that *underestimating the size of the interval* increases the error detection but quickly leads to a rise in the number of false positives due to the intrinsic difference between FP64 and FP32 mistakenly being flagged as an error. For $\delta \approx 1 \times 10^{-7}$, no more false positives occur, and about 75% of the errors in a single operation[1] can be detected (100% error detection is intrinsically impossible for RP-DWC, given the intrinsic floating point size mismatch). On the other hand, *overestimating the detection interval* leads only to a slow degradation in the detection rate.

For GPUs, we use one single *sub-optimal* EPL for all threads or for each block of threads (in cases where the code computation changes a lot from block to block, as in the case of particle simulations as LavaMD). To estimate the EPL we execute, off-line, the code with a set of inputs (that can be bound or not, random or not, depending on the structure and knowledge of the inputs) and choose as EPL the interval spanning the lowest and highest observed difference (δ_r or δ_{UINT}) measured for a code in the absence of faults. The use of a sub-optimal EPL for each different thread, as we show in Section VI, still allows a significant fault coverage.

[1]In this particular experiment, this curve showed the same behavior for the ADD and MUL operations.

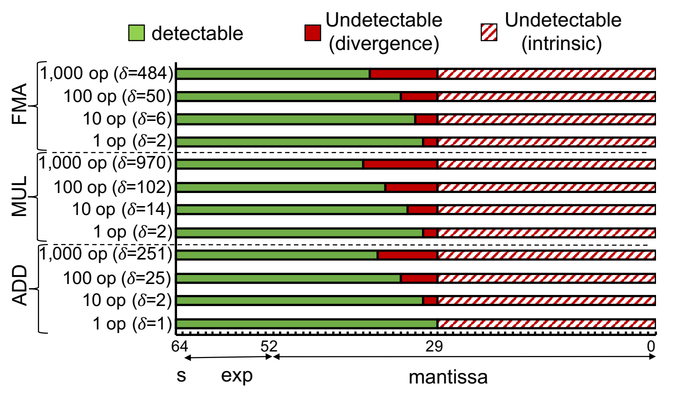


Fig. 4. Detectable and undetectable errors in the 64 bits of the original copy output based on EPL, which is reported as UINT value.

This process can be automated and integrated into the compile flow by testing the code for different ranges of input, and determining the EPL for each different range, which is later on applied at run time based on the real observed input range.

B. Detectable and Undetectable Errors

1) Undetectable Intrinsic Differences: There are two reasons why z_{64} and z_{32} will be different: *representation precision* and *computational divergence*. As the comparison is made in FP32 (with 23 mantissa, 8 exponent and 1 sign bits) while the result of the computation is in FP64 (with 52 mantissa, 11 exponent and 1 sign bits), faults in the 29 least significant bits of the FP64 mantissa will be *intrinsically* undetectable due to difference in representation precision.

In addition to that, we also need to consider the possible computational divergence. Based on approximate computing experience, the EPL can still be bound, thus allowing good fault detection. According to Goldberg [30], in the worst case of the execution of a single fault-free operation, $\delta_r < 2$ but, if a guard bit is used (or is intrinsically present), $\delta_r < 2\epsilon$, where ϵ is the *machine epsilon*, usually taken to be the largest relative roundoff error in that precision. The only known case that can exacerbate δ_r (and potentially jeopardize RP-DWC detection) is the *catastrophic cancelation* that occurs when two numbers are multiplied and then subtracted to a number with the same order of magnitude. In that case, the (small) δ_r of multiplication is expanded and becomes a (big) cancelation during subtraction. Catastrophic cancelations are rare and can be easily solved, at compiler time, by re-ordering operations [30]. In any case, as described in Section III-B, in Step 5 of RP-DWC includes a roll-back mechanism (FP64 - FP64 execution after an error detection) that prevents faults from remaining continuously generating false positives even in the case of catastrophic cancellation.

As an example to evaluate the dependence of EPL on the executed operation, granularity (length of operations sequence), and input values, we consider some instructions (ADD, MUL, FMA) and execute them in FP64 and FP32 with input restricted to various fixed ranges (0-10, 10-100, 100-1000, ...). Then, we relax the restriction on the input and use a wide range of values. We have run each configuration using more than 10M random values in the specified input ranges, without injecting faults, to measure the EPL. For a sequence of 10, 100, or 1,000 MUL, we do not consider those input ranges that overflow when running in FP32 (in the event of overflow RP-DWC would trigger the FP64-FP64 execution). EPL will be used in the error detection operation (using δ_r or *UINT*, as described in Step 4 of Section III-B).

To help to visualize the impact of the undetectable errors in the FP64 representation, in Figure 4, we consider the UINT difference δ_{UINT} (δ_r will have similar results but harder to visualize) for the selected configurations. We distinguish those bits of z_{64} that, even if corrupted, will not trigger error detection (in red or red dashed lines) from the bits that will be identified as corrupted in the event of an error (in green). The intrinsic precision difference bits (from bit 28 to 0) are marked as dashed red lines in Figure 4. Those are the precision bits of z_{64} that are not representable in z_{32}. The undetectable bits because of result divergence are marked in red. If EPL is set to $0 \leq \delta_{UINT} \leq 25$, for instance, we conservatively plot in red 5 additional bits (conservatively, $2^5 = 32$. Using only 4 additional bits might led to false positive) of z_{64} mantissa (bits from 33 to 29). The correspondent δ_r we measured ranges from 10^{-6} to 10^{-4}, depending on the granularity. A higher EPL will increase the number of digits that will not trigger error detection, moving the undetectable bits limit to the left side (most significant bits).

As shown in Figure 4, when only 1 operation is performed, EPL has the lowest values (2 in the worst case of MUL). When a longer sequence of operations is executed before error detection EPL increases, reaching 8 for ADD, 9 for FMA, and 10 for MUL. As shown in Figure 4, most of the 64 bits of the results, if corrupted, will not trigger error detection. However, errors in those bits that are more significant from computation are detectable. Additionally, an error during computation is likely to affect more than one bit of the output and will not be detected if all the corrupted bits are in the undetectable zone.

2) Limited Coverage of RP-DWC (FP Only): The fundamental idea of Reduced-Precision DWC makes it applicable only to instructions that perform floating-point numeric computations and when both inputs have a range of representation inside of that of the lower precision copy.

These two conditions only apparently undermine RP-DWC applicability. In fact, on the average, at least 50% of instructions of Rodinia benchmarks suite are floating point. Additionally, object-detection frameworks (YOLOv3 and YOLO-t) and particles simulations have more than 80% of floating-point instructions that the RP-DWC strategy can efficiently protect.

Previous work [31] has shown that most HPC applications use FP64 to have higher precision, not to increase the representation range of values, ensuring RP-DWC applicability. The rare event of having input out of FP32 range can be easily detected while performing the cast (as shown in Section III-B), forcing the use of traditional DWC for that specific operation.

V. Evaluation Methodology

In this section, we detail the architecture, benchmarks, and experimental procedure used to evaluate RP-DWC detection capabilities, performance and energy efficiency.

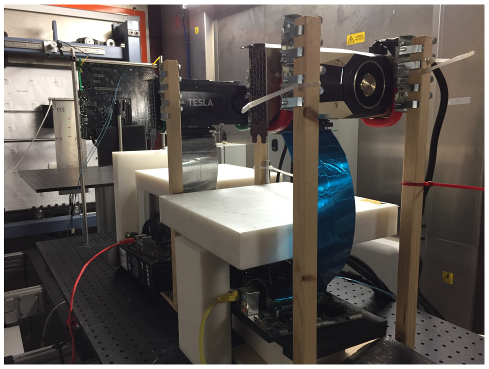


Fig. 5. Part of the experimental setup at ChipIR. The white blocks are boron plastic used to protect the motherboards from scattering neutrons.

A. Architectures and Benchmarks

The GPUs used for fault injection and beam experiments are NVIDIA Titan V, which implements the Volta microarchitecture. Volta GPUs feature dedicated hardware acceleration for three IEEE754 float point precisions: double, single, and half (i.e., FP64, FP32, and FP16, respectively). A thread can use a single-precision core to execute the same instruction in two half-precision operands in parallel [19]. In addition to that, we also experimented with the embedded Jetson Xavier as it includes dedicated hardware modules to measure the energy consumption (disabled by NVIDIA in the other Volta GPUs). The main difference between Xavier and Titan V is the number of available CUDA cores (512 vs. 5,120, respectively). We anticipate that as the structure of both GPUs is extremely regular, and the code that is run being the same, the observed relative overheads are very similar. No significant difference in the observed detection rates was found despite the difference in computational power.

For our evaluation, we design three Microbenchmarks (MUL, ADD, and FMA), which are a set of synthetic applications designed to stress specific components of the GPU architecture. Each Microbenchmark is composed of 20,480 threads (256x80, theoretical occupancy for an SM block scheduler x number of SMs), each performing a million dependent or independent arithmetic operations (multiplications, additions, or fused multiply-add). We have applied RP-DWC to these microbenchmarks at different granularities, i.e., performing the comparison every 1, 10, 100 or 1000 instructions as discussed in Section III-C). Additionally, we apply RP-DWC to four realistic codes: GEMM ($8k\times8k$), which is a cornerstone code for several applications (including convolutions) and performance evaluation tools, LavaMD (192 blocks), which simulates particle interactions in a large 3D space [32], Fast Wash Transform (FTW), that performs an orthogonal, symmetric, involutive, linear operation on 16M values, and BlackScholes, that uses partial differential equations on three arrays of 100k values, for predicting the dynamics of a financial market [33].

We use microbenchmarks to have a more controlled case-study to evaluate the impact of fine vs. coarse-grain RP-DWC and how detected and undetectable errors propagate through the code. Realistic applications are used to show a pragmatic evaluation of RP-DWC.

B. Fault-Injection and Neutron-Beam Experiments

We inject single bit faults using CAROL-FI [14], an open-source fault injector created with GDB (GNU Project Debugger) and Python. Among the available fault injectors, such as SASSIFI, FTAPE, Ferrari, and BIFIT [12], [13], we choose CAROL-FI because it supports novel NVIDIA CUDA platforms and is much faster than other GDB-based fault injectors [14]. CAROL-FI adds, on average $5\times$ overhead on application execution time. To evaluate the fault detection capabilities, our results are reported in terms of the Architectural Vulnerability Factor (AVF), which is the probability for a fault in a resource to affect the output [34]. We inject more than 2,000 faults for each configuration (sufficient to saturate the AVF and provide 95% confidence interval [13]), that is, more than 420,000 faults were injected.

Additionally, to evaluate the error detection capabilities of RP-DWC in real scenarios, we expose the Titan V executing the microbenchmarks to the accelerated neutron beam produced at the LANSCE facility in Los Alamos, NM, and at the ChipIR facility in Didcot, UK. Figure 5 shows part of the experimental setup mounted at ChipIR. Both these facilities provide a neutron spectrum of energy compatible with the terrestrial one, although accelerated [35], allowing to predict the error rates on a realistic application expressed in Failure In Time (FIT). The neutron flux available at LANSCE or ChipIR was between $1 \times 10^5 n/(cm^2 \times s)$ or $2.5 \times 10^6 n/(cm^2 \times s)$, about 6 to 8 orders of magnitude higher than the atmospheric neutron flux at sea level ($13n/(cm^2 \times h)$ [36]). Each configuration was tested for more than 100 hours of beam time. If scaled to the natural environment, this covers at least $\times 10^8$ hours of normal operations, which are about 11,000 years.

During fault injection and beam experiments, we let the execution complete even when an error is detected. This is necessary to observe the impact the detected error would have on the output and compare it with undetected errors.

VI. Experimental Evaluation

As described in Section IV-A, we choose as Expected Precision Loss (EPL) the lowest value that does not allow any false-positive detection executing the codes with random input without injecting faults. To evaluate the benefits and drawbacks of using RP-DWC on GPUs we compare its error coverage and overhead with traditional DWC (implemented duplicating all the instructions inside a thread and performing the correctness check just at the end, i.e. the coarser granularity, that guarantees the lowest overhead) and with state-of-the-art duplication strategies [5], [6].

We also measure the impact the detected and undetected errors would have in the application output, showing that the errors that RP-DWC cannot detect have a much lower impact on the output correctness than the ones it detects.

A. Fault-Injection Experiments

1) Architectural Vulneratibility Factor (AVF): We first analyze the AVF, i.e., the probability for an injected fault (random single bit flip) to propagate and generate a DUE (crash/hang) or a (detected or undetected) SDC. We present results using

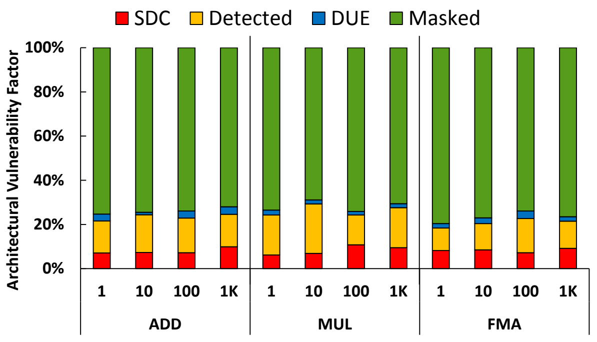


Fig. 6. AVF results for the microbenchmarks.

δ_{UINT} to compare the two copies. The use of δ_r would not affect the reported data significantly, having an impact on overhead lower than 5% in the fine-grain RP-DWC and negligible in the coarse-grain RP-DWC.

We plot, in Figure 6, the percentage of injected faults that are masked, that became an SDC that is detected or undetected, or that produce a DUE. We plot data obtained for the microbenchmarks with different RP-DWC granularities, that is, performing the correctness check at each instruction, every 10, 100, or 1,000 instructions. Figure 6 demonstrates that most of the injected faults are either masked or detected. Additionally, the AVF for DUE is very small compared to the AVF for SDC. This is because DUEs are mainly produced by errors in control logic or interfaces [15], [16], while our injections target mostly the datapath. Figure 6 also shows that there is no significant difference in the AVF between operations and configurations (fine or coarse grain RP-DWC). Having similar AVF attests that the configurations and operations have a similar probability for a fault to propagate or to be masked. Fault injection does not include any information about the probability for a fault to happen during the different computations. In other words, we cannot evaluate from Figure 6 alone if the additional instructions required to implement duplication have an effect on the code error rate. Beam experiment data, presented in the next subsection, will also consider this aspect.

Table II provides additional details on the efficiency and efficacy of RP-DWC by listing the percentage of errors that are detected and the imposed overheads (execution time, power and energy consumption). We also include the detection rate and the overheads of a traditional DWC to ease the comparison with RP-DWC. Please recall that, while for RP-DWC microbenchmarks, we show data for different granularities, for the traditional DWC and for the realistic codes (GEMM and LavaMD) we show only the result for the duplication of the entire code, which is the best case scenario for the overheads.

2) Error Detection: As shown in Table II, the percentage of detected errors for RP-DWC ranges from 57% for 1,000 op FMA to 76% for 10 op MUL. While increasing the granularity increases the threshold (details in Section IV), there seems to be no correlation between the detection rate and the granularity. As discussed in Section III-C, there are two reasons for that: (1) Fine-grain RP-DWC requires a check at every executed instruction, increasing the probability of having an (undetectable) check operation corruption. (2) An undetectable error in the first operations of a block in a coarse-grain RP-DWC can increase in magnitude as it propagates, eventually becoming detectable when the correctness check is performed. The longer the block of instruction, the higher the probability for the undetectable errors to become detectable. That implies a tradeoff is present between the performance overhead and the time to detection, but not between the performance and the detection rate.

TABLE II
ERROR DETECTION AND OVERHEAD (FAULT INJECTION).

Benchmark	Granularity	Overhead Time	Overhead Power	Overhead Energy	Detection
ADD (RP-DWC)	1 op	34,9%	51,2%	104,0%	67.1%
	10 op	3,9%	35,7%	41,0%	70.1%
	100 op	0,3%	27,6%	28,0%	68.6%
	1,000 op	0,1%	23,9%	24,0%	59.8%
MUL (RP-DWC)	1 op	35,3%	52,3%	106,0%	74.5%
	10 op	3,6%	36,1%	41,0%	76.5%
	100 op	0,3%	32,6%	33,0%	59.3%
	1,000 op	0,1%	31,9%	32,0%	70,0%
FMA (RP-DWC)	1 op	35,3%	55,9%	111,0%	55.4%
	10 op	3,6%	38,0%	43,0%	58.3%
	100 op	0,3%	34,6%	35,0%	68.3%
	1,000 op	0,1%	32,9%	33,0%	57.2%
GEMM (RP-DWC)		13,0%	43,4%	62,0%	63.4%
LavaMD (RP-DWC)		18,1%	32,1%	56,0%	83.4%
BlackScholes (RP-DWC)		5,8%	7,5%	13,8%	66.2%
FWT (RP-DWC)		37,4%	9,2%	50,1%	75.6%
ADD (trad. DWC)		94,2%	8,7%	111,0%	>95%
MUL (trad. DWC)		94,4%	15,2%	124,0%	>95%
FMA (trad. DWC)		99,1%	12,5%	124,0%	>95%
GEMM (trad. DWC)		70,3%	22,1%	108,0%	>95%
LavaMD (trad. DWC)		83,6%	12,7%	107,0%	>95%
BlackScholes (trad. DWC)		50,5%	17,5%	76,7%	>95%
FWT (trad. DWC)		94,6%	6,2%	106,6%	>95%

The lower detection rate compared to traditional DWC (over 95% from Table II) or state-of-the-art DWC from previous work (80%-95% as listed in Table I) is not surprising, since, as shown in Section IV-B, any faults hitting or propagating to the less-significant bits of an FP64 number are not detectable. Even though this limitation of RP-DWC provides an upper bound of erroneous bits coverage it can achieve, these wrong least significant bits are exactly the ones that will provide a smaller impact to the application output, as we show in Section VI-C. RP-DWC has a higher detection capability than previous work that approximates the algorithm or proposes approximated hardware for error detection ([8], [10], [28]), which is, as listed in Table I, 55%-76% for RP-DWC and 20%-40% for previous work. This data attests that approximating the algorithm might not be as effective as approximating the hardware. The higher error detection of RP-DWC compared to the use of dedicated approximated hardware could be caused by the higher approximation chosen in [7], [8] and by more reliable hardware designed by NVIDIA.

3) Overhead: Data in Table II shows that the execution time overhead of RP-DWC (35% in the worst case of a fine-grain RP-DWC) is much lower than traditional DWC (70%-90%) and of recent efficient DWC (39% in [6]). Our implementation of traditional DWC has a lower overhead compared to some previous studies that showed an overhead of 2x [4], as we duplicate operations inside a thread rather

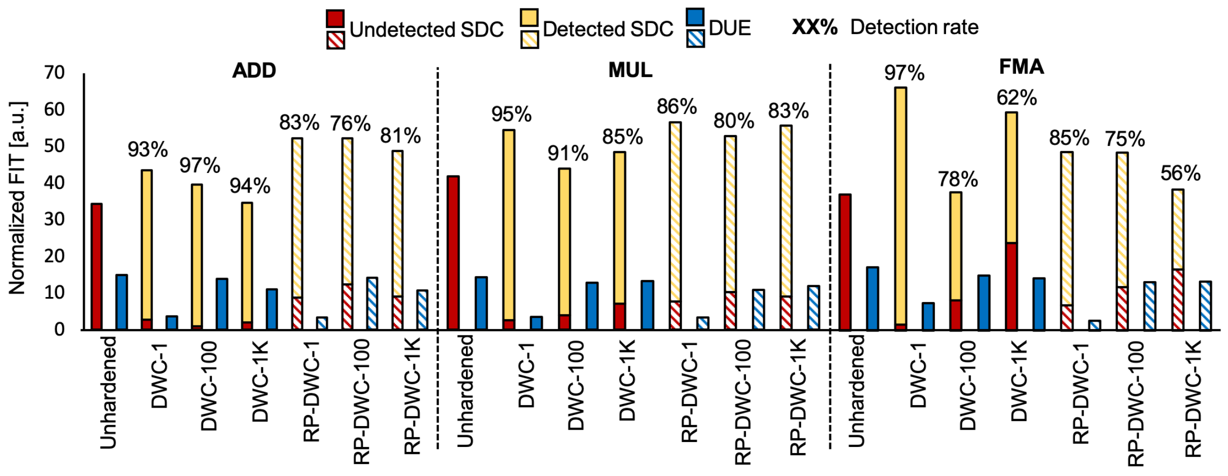


Fig. 7. Normalized beam experimental data for the unhardened, the traditional DWC, and RP-DWC versions of the microbenchmarks. Dashed lines are used just to highlight the RP-DWC versions.

than threads or blocks of threads. If FP64 cores are available, the GPU could then be able to schedule some of the two FP64 copies in parallel. As expected, as we reduce the granularity of the correctness check, the overhead is reduced. When we increase the granularity, as the comparison operations are less frequent, the overhead is reduced accordingly, eventually becoming nearly-zero.

The Instructions Per Cycle (IPC) and the number of executed instructions of the RP-DWC version are approx. 2x the unhardened versions, for all configurations. On the contrary, for the traditional DWC versions, while the number of executed instructions is slightly higher than 2x, the IPC is very similar to the unhardened version (lower than 1.2x). This also proves that the available resources now contribute, with a nearly zero overhead, to the reliability improvement.

To further investigate the overhead differences between RP-DWC and traditional DWC, we consider the pressure on the register file. The higher register file usage is found for GEMM traditional DWC, for which 41 registers per thread are instantiated. As the register limit on NVIDIA Volta is 256 per thread, none of the configurations we tested saturates the register file. The higher overhead for traditional DWC, then, comes from the saturation of computing units.

Table II shows that the energy overhead of RP-DWC is significantly reduced when the correctness checks are less frequent, reaching a value that can be as low as 24% to 32% BlackScholes reaches an even lower energy overhead (13.8%). However, such a low overhead is not solely justified by RP-DWC, but also by the simplicity of the code (few global memory access, no shared memory utilization, and executes only simple operations) [33]. For the same reasons, the energy overhead of the traditional DWC is lower than for the other codes. The energy consumption overhead of RP-DWC is comparable to the traditional DWC only for the fine-grain implementation (1 check every operation for RP-DWC vs. one check at the end of the application for traditional DWC). The energy consumption overhead of a fine grain RP-DWC is, then, higher than 100%. This is justified because we are actually executing 3x the instructions of the unhardened version (FP64 and FP32 copies plus the error detection operation). Traditional DWC has, even with the check only at the end of the computation, a higher energy consumption overhead, which ranges from 111% to 124%. This favorable energy consumption result of RP-DWC is achieved by leveraging the FP32 cores to execute the redundant copy in parallel.

Finally, for RP-DWC the energy consumption overhead is always higher than the time overhead as the error detection operation (UINT or δ_r) can be done in parallel with floating-point operations in GPUs, reducing the time but not the energy overhead of activating the idle cores for error detection. To justify that, Table II also provides numbers on the power consumption of each application and duplication approach. These results suggest that, when activated for the RP-DWC, the FP32 cores can contribute to 15-20% of the total power consumption depending on the ratio of static/leakage to dynamic power (a parameter that can be tuned in the manufacturing process according to the target domain). Since this power overhead is still substantially lower than the $\sim$2x speedup achieved from using RP-DWC instead of DWC, the energy consumption of RP-DWC is also lower. It should be noted that the FP32 cores are not power gated when only FP64 is being used (i.e, in the case of traditional DWC) because fine-grained power gating at the level of individual FP32 units typically requires complex circuit-level support to be realized and is not available in COTS devices such as the one we evaluate. However, even if the static power consumption is set to 30% of the total power (a high estimate, even for current process nodes), power-gating FP32 cores when executing FP64 applications would only reduce power consumption by an estimated 12%, a reduction still much lower than performance improvements of RP-DWC compared to DWC.

B. Neutron Beam Experiments

To have an even more realistic evaluation of the effectiveness of RP-DWC and a direct comparison with traditional DWC, we expose the GPUs to accelerated neutron beams. Dissimilarly to fault injection, during beam experiments, all the GPU resources are irradiated and could be corrupted, and the fault model is as close as possible to the real one.

In Figure 7, we report the beam experiment results for the microbenchmark in the unhardened version (no duplication), protected with a traditional DWC, and with RP-DWC, in three different granularities (checking after 1, 100, or 1,000 operations). The detection rates of DWC and RP-DWC are made explicit in the Figure to ease comparing the effectiveness of the two techniques. The reported values are affected by a 15% experimental error due to statistic, and neutrons count uncertainty. FIT rates are normalized to the lowest measured value (DUE rate of DWC-1 for ADD) not to reveal business-sensitive data but still to allow a direct comparison between configurations. The dashed lines are used just to differentiate the RP-DWC results from the traditional DWC ones.

From Figure 7, we notice that the SDC FIT rate for the unhardened version is lower than the FIT rate of the protected versions (considering the combination of detected and undetected SDCs). This is expected, as the check operation introduces a computation and memory overhead that have the side effect of increasing the protected versions error rate. The increased FIT rate is higher when the check is performed at each instruction, as more instructions are being executed. RP-DWC has a higher increase in the SDC rate for all configurations but FMA. This is because the cast and error detection operations (detailed in Section III-B) are computationally more costly than ADD and MUL, but not of FMA. Nevertheless, both traditional and RP-DWC detects most of the SDCs, resulting in a much lower undetected SDC rate than the unhardened versions.

As observed with fault injection, and for the same reason, the detection rate of the traditional DWC is always higher than the RP-DWC. On average, under the beam, the detection rate of RP-DWC is 9% lower than DWC. The fact that the detection rates of RP-DWC are higher than those reported in Table II should not surprise. As mentioned in Section V, data in Table II is obtained injecting only single bit flips, which are the harder to detect with RP-DWC. In fact, when even one corrupted bit is outside of EPL, RP-DWC triggers the detection. The higher the number of bits flipped, the higher the probability of detection. As already studied in previous work [4], [37], radiation may induce complex fault models in GPUs (multiple bit flips, the corruption of several threads, and so on) that are easier to detect with RP-DWC. Detection rates in Table II are then a conservative estimation of RP-DWC detection capabilities. Nevertheless, radiation during beam experiment can also corrupt data before it is assigned to the two copies and, thus, resulting in undetected errors. The presence of ECC would remove this possibility (the tested Titan V does not have ECC), possibly improving further the detection rates of both DWC and RP-DWC.

The use of RP-DWC, then, results in a slightly lower error detection but makes both overheads much smaller than a traditional DWC and also of state-of-the-art DWC for GPUs as reported in [5], [6]. As shown in the next subsection, the errors RP-DWC does not detect have a minimal impact on the output correctness and could potentially be tolerated, mitigating the impact of the observed lower detection.

C. Detected vs Undetected Errors

As discussed in Section III-B, in a real application of RP-DWC, once an error is detected, the computation would either be terminated or the roll-back mechanism would be triggered. In our experiments, we allow the execution to complete even if an error is detected in order to compare the impact of detected and undetected errors in the output correctness. To measure the impact of these errors, we use the concept of *Tolerated Relative Error (TRE)* as introduced in [11]. A TRE of 0% implies no tolerance in the output correctness, i.e., the computed output must match the expected output. In other words, any mismatch between the computed output and the expected value is considered a critical error. Increasing the TRE relaxes the correctness constraint accepting (corrupted) values in a given range as tolerable (corrected). As an example, if we consider a TRE of 10%, any output value between 90% and 110% of the expected value will be considered as correct or, at least, tolerable. If the output is composed of multiple elements, as in the case of out tests, *all elements* must have a tolerable value for the execution not to be considered corrupted.

In Figures 8 and 9, we show how the detected and undetected errors affect the code output. Only fault-injection data is reported, beam experiment results are similar and not shown. We plot, in the y-axis, how much the SDC AVF rate would be reduced as a function of TRE (that varies from 0% to 1% in the x-axis). When TRE is 0% the AVF will be exactly the experimental one (100%). As we increase the TRE, some corrupted executions eventually become tolerable and, then, the AVF will be reduced accordingly. The fact that a small TRE is sufficient to significantly reduce the AVF indicates that most of the errors have a small impact on the output value.

Data in Figures 8 and 9 demonstrates that, for all codes, the critical error rate reduction is much faster for the undetected faults (dashed lines in Figures 8 and 9). This indicates that undetected faults have a much lower impact on the output correctness than the impact the detected errors would have. Even with a TRE as small as 0.1% all the undetected errors would be tolerated for all the microbenchmarks (but MUL with 100 ops) while at least 60% of the detected errors would still be considered critical. This means that the errors RP-DWC is unable to detect have an impact on the output value that is lower than 0.1%. For MUL, as said, we need to use a smaller input to avoid overflow in the FP32 replica. A small error might then induce a higher TRE. However, with a TRE of less than 5% (not shown), all the undetected errors for MUL would be considered tolerable. For GEMM and LavaMD, shown in Figure 9, the trend is maintained, but the impact of undetected faults is slightly higher ($\sim$90% of the undetected errors modify the output of less than 1%). This is because GEMM and LavaMD include several instructions of different nature (i.e., ADD, MUL, FMA, and other instructions). It is worth noting that in a traditional DWC (not shown) the detected and undetected error effects in the application output overlap as the detection is independent of the relative value of the error.

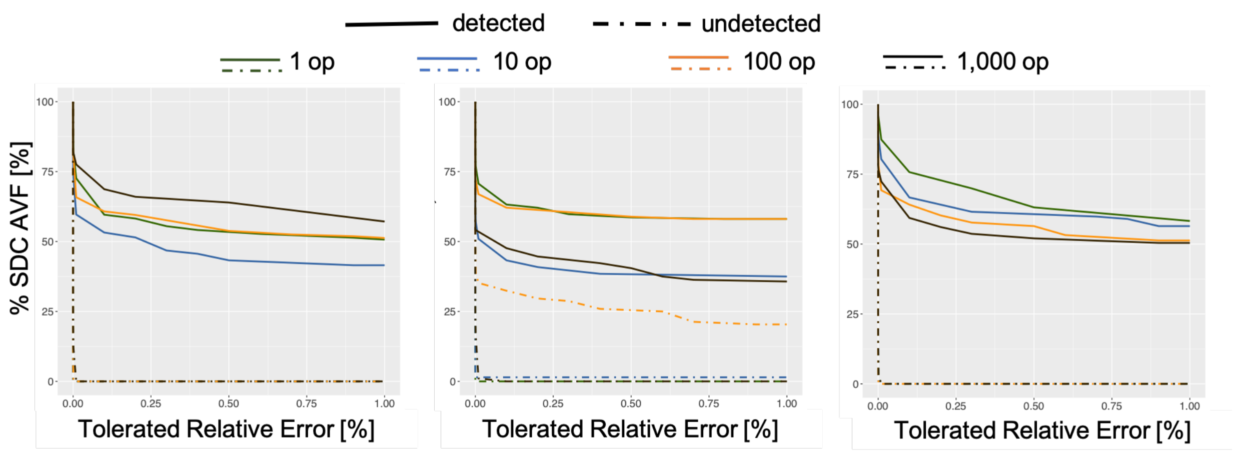


Fig. 8. Impact of detected and undetected errors in the output correctness for the microbenchmarks.

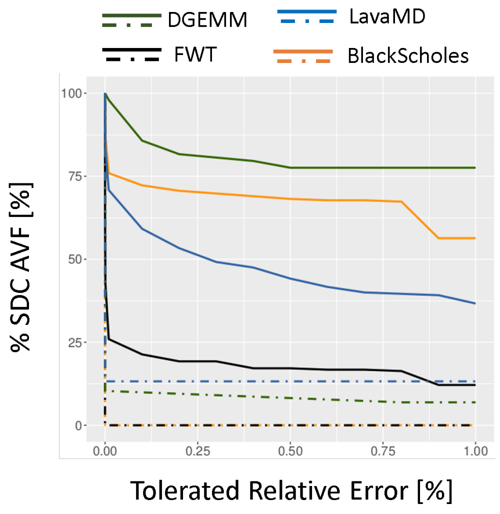


Fig. 9. Impact of detected and undetected errors in the output correctness for GEMM, LavaMD, FWT, and BlackScholes.

D. Impact of Undetected Errors in HPC and Safety-Critical Applications

While both the detected and undetected errors are to be considered different from the expected FP64 result, data reported in Figures 8 and 9 shows that our technique detects those errors that are more likely to generate a significant deviation from the application output. Previous works have demonstrated that small fluctuations in the output value (up to 4%) have a low or negligible impact and can, potentially, be even tolerated for several HPC applications such as particles or physical simulation, weather forecast, heat distribution, wave propagation, earthquakes prediction, etc... [22]. The realistic codes we tested (GEMM, LavaMD, FWT) are among these HPC applications. As shown in Table II and Figure 7, RP-DWC, while being more efficient, has a lower error detection rate than traditional DWC (of ∼15% for fault injection and ∼9% for beam experiments). Nevertheless, as shown in Figures 8 and 9, even considering a fluctuation in the output value (i.e., a TRE) of 1%, more than 90% of the undetected errors with RP-DWC are considered tolerable (99% for the micro-benchmarks). As a result, almost all the RP-DWC undetectable errors, according to [22], can be tolerated in several HPC applications.

Even for some safety-critical applications, such as Convolutional Neural Networks (CNNs) for objects detection, small variations in the output values are unlikely to induce critical faults [38], [39]. A fault is critical for a CNN when it induces a misdetection. On the other hand, a fault that only slightly modifies the CNN output but *does not* impact detection will not cause vehicle misbehaviors and, thus, can be considered tolerable We run an additional fault injection campaign to evaluate the percentage of detected vs undetected errors that are critical for CNNs. We will focus on RP-DWC for GEMM, as more than 70% of operation in a CNN are related to matrix multiplications and, as shown in [38], most of CNN errors are caused by GEMM corruptions. To understand if the undetected errors would cause critical errors for a CNN, we perform a fault injection campaign on YOLOv3 processing frames from VOC2012 dataset [40]. We modify the output of a random matrix in a random layer of YOLOv3. We first inject a relative error of 1% as, according to Figure 9, the vast majority (∼90%) of GEMM errors RP-DWC is unable to detect have a TRE lower than 1%. To consider the worst-case scenario for RP-DWC, we corrupt *all the elements* or *a row of elements* of the matrix. *None* of the 1,000 injections induced misdetections in YOLOv3. This result confirms that the faults RP-DWC is unable to detect have a negligible impact even in CNNs, which is not sufficient to cause misdetections. Then, we inject with a relative error in the range 1% to 100% and about 12% of these errors (i.e., the ones RP-DWC is highly likely to detect) induced misdetections.

We can derive that, while RP-DWC has a lower detection rate than a traditional DWC, only ∼10% of the errors RP-DWC is unable to detect (i.e., the ones with a TRE higher than 1%) can potentially be critical for HPC or safety-critical applications. If we consider that the other ∼90% of undetectable errors are actually tolerable errors, then the detection rate of RP-DWC increases from the 55%-83% (see Table II) to 95%-98%, which is comparable with the traditional DWC. The decision of considering or not the tolerable errors, as well as the tolerance threshold, depends on the application. In any case, RP-DWC shows a better cost-benefit trade-off (overhead-detection) than traditional DWC, and could be particularly useful for HPC applications.

E. Projections for Different Architectures

The proposed RP-DWC evaluation is based on NVIDIA GPUs with redundant mixed-precision hardware, which exacerbates the overhead reduction brought by a lower precision redundant copy. Nevertheless, the same implementation and discussion about RP-DWC we provide can also be applied to parallel or sequential devices in which the same hardware is shared among different precisions (as is the case for Intel x86 architecture). In these architectures, the redundant copy will be executed *sequentially* rather than in parallel with the original code, increasing the overhead of duplication (traditional or RP). RP-DWC, however, will still guarantee a lower overhead compared to a traditional DWC. The overhead reduction will be directly proportional to the efficiency of executing the reduced precision operation compared to full precision. Previous work showed that, even in the absence of dedicated hardware, the execution of an FP32 instruction requires half the time and energy of an FP64 instruction [19]. We can then expect that duplicating an FP64 code with an FP32 replica will impose, on the average, about half the overhead of a traditional DWC. There is no reason to think that the difference in detection between DWC and RP-DWC that we observe for GPUs wouldn't also hold for other architectures, including the fact that the undetected faults are also the ones that lead to small magnitude errors.[2] A transient fault, in fact, will affect only the copy executing while the fault is active. When the next copy runs in the previously corrupted hardware, it will use new data (the copies are independent) overriding the fault.

VII. Conclusions

In this paper, we have presented Reduced-Precision DWC, a strategy to detect errors duplicating the original instructions with reduced-precision instructions. The technique is particularly promising for (but not limited to) mixed-precision architectures, as novel GPUs that have dedicated hardware cores for the execution of different precision operations. RP-DWC, then, uses available hardware, that would other otherwise be idle, to improve the reliability of codes with reduced overhead.

Among the limitations of RP-DWC, the most challenging one is the reduced error detection capability that comes from the different intrinsic results of different precision executions. However, we have shown that a significant amount of errors can still be detected and, more importantly, the undetected errors are the ones that have a smaller impact on the output correctness. As a result, undetected errors may still be tolerated by various applications.

VIII. Acknowledgments

This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Sklodowska-Curie grant agreement No 886202 and from The Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brazil (CAPES) - Finance Code 001. Neutron beam time was provided by ChipIR (DOI: 10.5286/ISIS.E.RB2000161) thanks to C. Cazzaniga and C. Frost and by LANSCE thanks to Steve Wender and Gus Sinnis. Authors would also like to thank Rubens Luiz Rech Junior for his precious help.

[2]Assuming very similar EPL.

References

[1] R. Lucas, "Top ten exascale research challenges," in *DOE ASCAC Subcommittee Report*, 2014.

[2] J. Dongarra, H. Meuer, and E. Strohmaier, "ISO26262 Standard," 2015.

[3] R. Baumann, "Radiation-induced soft errors in advanced semiconductor technologies," *Device and Materials Reliability, IEEE Transactions on*, vol. 5, pp. 305–316, Sept 2005.

[4] D. A. G. de Oliveira, L. L. Pilla, T. Santini, and P. Rech, "Evaluation and mitigation of radiation-induced soft errors in graphics processing units," *IEEE Transactions on Computers*, vol. 65, pp. 791–804, March 2016.

[5] J. Wadden, A. Lyashevsky, S. Gurumurthi, V. Sridharan, and K. Skadron, "Real-world design and evaluation of compiler-managed GPU redundant multithreading," in *2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*, pp. 73–84, June 2014.

[6] A. Mahmoud, S. K. S. Hari, M. B. Sullivan, T. Tsai, and S. W. Keckler, "Optimizing software-directed instruction replication for GPU error detection," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis*, SC '18, (Piscataway, NJ, USA), pp. 67:1–67:12, IEEE Press, 2018.

[7] K. Seetharam, L. C. T. Keh, R. Nathan, and D. J. Sorin, "Applying reduced precision arithmetic to detect errors in floating point multiplication," in *2013 IEEE 19th Pacific Rim International Symposium on Dependable Computing*, pp. 232–235, Dec 2013.

[8] M. Maniatakos, Y. Makris, P. Kudva, and B. Fleischer, "Exponent monitoring for low-cost concurrent error detection in FPU control logic," in *29th VLSI Test Symposium*, pp. 235–240, May 2011.

[9] M. B. Sullivan, *Low-cost duplication for separable error detection in computer arithmetic*. PhD thesis, The University of Texas at Austin, 2015.

[10] G. Rodrigues, J. Fonseca, F. Kastensmidt, V. Pouget, A. Bosio, and S. Hamdioui, "Approximate TMR based on successive approximation and loop perforation in microprocessors," *Microelectronics Reliability*, vol. 100-101, p. 113385, 2019.

[11] F. Fernandes dos Santos, C. Lunardi, D. Oliveira, F. Libano, and P. Rech, "Reliability evaluation of mixed-precision architectures," in *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 238–249, Feb 2019.

[12] B. Fang *et al.*, "GPU-Qin: A methodology for evaluating the error resilience of GPGPU applications," in *Performance Analysis of Systems and Software (ISPASS), 2014 IEEE Int. Symposium on*, pp. 221–230, March 2014.

[13] S. K. S. Hari *et al.*, "SASSIFI: An architecture-level fault injection tool for GPU application resilience evaluation," *Int. Symposium on Performance Analysis of Systems and Software*, Oct 2017.

[14] D. Oliveira *et al.*, "Experimental and analytical study of xeon phi reliability," in *Proceedings of the Int. Conference for High Performance Computing, Networking, Storage and Analysis*, SC '17, (New York, NY, USA), pp. 28:1–28:12, ACM, 2017.

[15] V. Fratin *et al.*, "Code-dependent and architecture-dependent reliability behaviors," in *2018 48th Annual IEEE/IFIP Int. Conference on Dependable Systems and Networks (DSN)*, pp. 13–26, June 2018.

[16] A. Chatzidimitriou, P. Bodmann, G. Papadimitriou, D. Gizopoulos, and P. Rech, "Demystifying soft error assessment strategies on ARM CPUs: Microarchitectural fault injection vs. neutron beam experiments," in *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pp. 26–38, June 2019.

[17] M. Traiola, A. Savino, M. Barbareschi, S. D. Carlo, and A. Bosio, "Predicting the impact of functional approximation: from component-to application-level," in *IOLTS*, pp. 61–64, July 2018.

[18] P. Kudva, B. M. Fleischer, Y. Makris, and M. Maniatakos, "Low-cost concurrent error detection for floating-point unit (FPU) controllers," *IEEE Transactions on Computers*, vol. 62, pp. 1376–1388, jul 2013.

[19] NVIDIA, "NVIDIA Tesla V100 GPU architecture - whitepaper," Tech. Rep. 1.1, NVIDIA, aug 2017.

[20] S. Mittal, "A survey of techniques for approximate computing," *ACM Computing Surveys (CSUR)*, vol. 48, no. 4, p. 62, 2016.

[21] S. T. Chakradhar and A. Raghunathan, "Best-effort computing: Re-thinking parallel software and hardware," in *Design Automation Conference*, pp. 865–870, IEEE, 2010.

[22] J. de la Puente *et al.*, "Mimetic seismic wave modeling including topography on deformed staggered grids," *GEOPHYSICS*, vol. 79, no. 3, pp. T125–T141, 2014.

[23] S. Gupta *et al.*, "Deep learning with limited numerical precision," in *Proceedings of the 32Nd Int. Conference on Machine Learning - Volume 37*, ICML'15, pp. 1737–1746, JMLR.org, 2015.

[24] IEEE, "IEEE standard for floating-point arithmetic," *IEEE Std 754-2008*, pp. 1–70, Aug 2008.

[25] R. Krashinsky, O. Giroux, S. Jones, N. Stam, and S. Ramaswamy, "NVIDIA Ampere Architecture In-Depth." NVIDIA Developer Blog, 2020.

[26] Byonghyo Shim, S. R. Sridhara, and N. R. Shanbhag, "Reliable low-power digital signal processing via reduced precision redundancy," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 12, no. 5, pp. 497–510, 2004.

[27] M. Biasielli, L. Cassano, and A. Miele, "An approximation-based fault detection scheme for image processing applications," in *2020 Design, Automation Test in Europe Conference Exhibition (DATE)*, pp. 1331–1334, 2020.

[28] Y. Zhang, R. Nathan, and D. J. Sorin, "Reduced precision checking to detect errors in floating point arithmetic," *CoRR*, vol. abs/1510.01145, 2015.

[29] G. Chakrabarti, V. Grover, B. Aarts, X. Kong, M. Kudlur, Y. Lin, J. Marathe, M. Murphy, and J.-Z. Wang, "CUDA: Compiling and optimizing for a GPU platform," *Procedia Computer Science*, vol. 9, pp. 1910–1919, 12 2012.

[30] D. Goldberg, "What every computer scientist should know about floating-point arithmetic," *ACM Comput. Surv.*, vol. 23, pp. 5–48, Mar. 1991.

[31] J. Langou, J. Langou, P. Luszczek, J. Kurzak, A. Buttari, and J. Dongarra, "Exploiting the performance of 32 bit floating point arithmetic in obtaining 64 bit accuracy (revisiting iterative refinement for linear systems)," in *SC '06: Proceedings of the 2006 ACM/IEEE Conference on Supercomputing*, pp. 50–50, Nov 2006.

[32] S. Che *et al.*, "Rodinia: A benchmark suite for heterogeneous computing," in *Proceedings of the IEEE Int. Symposium on Workload Characterization (IISWC)*, pp. 44–54, 2009.

[33] A. Yazdanbakhsh, D. Mahajan, H. Esmaeilzadeh, and P. Lotfi-Kamran, "Axbench: A multiplatform benchmark suite for approximate computing," *IEEE Design Test*, vol. 34, no. 2, pp. 60–68, 2017.

[34] S. S. Mukherjee *et al.*, "A Systematic Methodology to Compute the Architectural Vulnerability Factors for a High-Performance Microprocessor," in *Proceedings of the 36th Annual IEEE/ACM Int. Symposium on Microarchitecture*, (Washington, DC, USA), pp. 29–, IEEE Computer Society, 2003.

[35] C. Cazzaniga and C. D. Frost, "Progress of the scientific commissioning of a fast neutron beamline for chip irradiation," *Journal of Physics: Conference Series*, vol. 1021, p. 012037, may 2018.

[36] JEDEC, "Measurement and Reporting of Alpha Particle and Terrestrial Cosmic Ray-Induced Soft Errors in Semiconductor Devices," Tech. Rep. JESD89A, JEDEC Standard, 2006.

[37] P. Rech *et al.*, "Impact of GPUs Parallelism Management on Safety-Critical and HPC Applications Reliability," in *IEEE Int. Conference on Dependable Systems and Networks (DSN 2014)*, (Atlanta, USA), 2014.

[38] F. F. d. Santos, P. F. Pimenta, C. Lunardi, L. Draghetti, L. Carro, D. Kaeli, and P. Rech, "Analyzing and increasing the reliability of convolutional neural networks on GPUs," *IEEE Transactions on Reliability*, vol. 68, no. 2, pp. 663–677, 2019.

[39] G. Li, S. K. S. Hari, M. Sullivan, T. Tsai, K. Pattabiraman, J. Emer, and S. W. Keckler, "Understanding error propagation in deep learning neural network (DNN) accelerators and applications," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '17, (New York, NY, USA), Association for Computing Machinery, 2017.

[40] J. Redmon and A. Farhadi, "YOLOv3: An incremental improvement," *CoRR*, vol. abs/1804.02767, 2018.

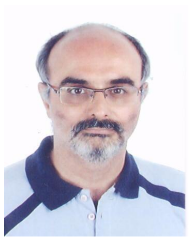
Fernando Fernandes dos Santos received his MSc degree from Universidade Federal do Rio Grande do Sul (UFRGS) in 2017 and his BSc from Universidade Estadual do Oeste do Paraná (UNIOESTE) in 2014. Currently, he is a Ph.D. student at UFRGS working on fault tolerance in HPC and safety-critical applications

Marcelo Brandalero received his Dr. Degree in Computer Science from Universidade Federal do Rio Grande do Sul (UFRGS) in Porto Alegre, Brazil in 2019, and is currently Senior Research Scientist in the Brandenburg University of Technology Cottbus-Senftenberg, in Germany. His research interests cover computer architecture with emphasis on reconfigurable architectures.

Michael B. Sullivan is a senior research scientist in the architecture research group at NVIDIA. His main research is the design of efficient, dependable, and secure large-scale computer systems. He received Ph.D and M.S. degrees in computer architecture from the University of Texas at Austin.

Pedro Martins Basso is currently a double degree Computer Engineering student at Universidade Federal do Rio Grande do Sul (UFRGS), Brazil and École Polytech Montpellier, France. He is currently working on radiation-induced effects in HPC systems and safety-critical applications, mainly on GPUs.

Michael Huebner is a Full Professor at the Brandenburg University of Technology, Cottbus, Germany, and the Chair of Computer Engineering Group. He received his habil. and Dr.-Ing. Degrees from the Karlsruhe Institute of Technology (KIT) in 2011 and 2007, respectively. His research interests are in reconfigurable computing with application in automotive systems.



Luigi Carro received the Electrical Engineering and the MSc degrees from Universidade Federal do Rio Grande do Sul (UFRGS), Brazil, in 1985 and 1989, respectively. In 1996 he received the Dr. degree in Computer Science at UFRGS, Brazil. He is presently a full professor at the Informatics Institute of UFRGS, in charge of Computer Architecture and Organization.

Paolo Rech received his master and Ph.D. degrees from Padova University, Padova, Italy, in 2006 and 2009, respectively. He is an associate professor at UFRGS in Brazil and a Marie Curie Fellow at Politecnico di Torino, Italy. His main research interests include the reliability of radiation-induced effects in large-scale HPC, autonomous vehicles and space exploration.