Software-only Triple Diverse Redundancy on GPUs for Autonomous Driving Platforms

Sergi Alcaide[‡,†], Leonidas Kosmidis[†,‡], Carles Hernandez[⋆,†], Jaume Abella[†]
[†]Barcelona Supercomputing Center (BSC)
[‡] Universitat Politècnica de Catalunya (UPC) [⋆] Universitat Politècnica de Valencia (UPV)

***Abstract*—Autonomous driving (AD) imposes the need for safe computations in high-performance computing (HPC) components such as GPUs, thus with capabilities to detect *and recover* from errors since a safe state may not exist anymore. This can be achieved with Triple Modular Redundancy (TMR) for computation components. Furthermore, error detection capabilities need to provide some form of diversity to avoid the case where a single fault leads all redundant executions lead to the same error, which would go undetected. In our past work, we assessed GPUs against dual modular redundancy (DMR) with diversity, showing their potential and limitations to provide diverse redundancy building on reset and restart for recovery. However, such recovery scheme may be too slow for some applications.**

This paper proposes a software-only solution to deliver diverse TMR on commercial off-the-shelf (COTS) GPUs. Our work details how staggered execution can be achieved and assesses the performance of TMR on COTS GPUs. Moreover, we identify those elements where diversity cannot be guaranteed and provide some discussion comparing the case of DMR and TMR for those elements.

I. Introduction

The humongous computation requirements of autonomous driving (AD) frameworks call for the use of high-performance computing (HPC) devices in future automotive platforms. Those HPC platforms, such as GPUs executing machine learning workloads for perception [1], have already been proven to deliver the performance needed by AD systems – as shown in recent AD systems demonstrations [2]. However, it is unclear how these platforms will adhere to the highest Automotive Safety Integrity Levels (ASIL) as dictated by the automotive functional safety standards, i.e. ISO26262 [3].

In the context of ISO26262, safety measures need to be deployed, and Validation and Verification (V&V) processes must be followed in order to collect evidence to prove that safety requirements are met even in the most stringer circumstances [4]. The level of assurance needed, provided that testing is simply unable to cover all possible situations, varies in accordance with the ASIL of the target application.

Automotive systems usually have a safe state upon a failure, either by stopping the car or giving the control to the driver to manage exceptional situations. This property allows relieving HPC components from any safety requirement as long as other components monitor the safe behavior of both, HPC components as well as themselves. Those safety-related monitors, therefore, inherit all safety requirements of the functionality delivered jointly by themselves and the HPC components. Upon an error in the HPC components, those monitors only need to guarantee the timely detection of the error, and transferring the system to a safe state within the fault-tolerant time interval (FTTI), thus transforming errors in HPC components into an availability concern rather than a safety one.

The highest integrity levels (i.e. ASIL-C/D) require avoiding *common cause failures (CCFs)*. A CCF is a failure caused by a single fault affecting all redundant components. Avoiding CCFs implies the use of diverse redundant designs so that either the fault cannot affect all redundant components or, if it may bring all them to error, such error manifests heterogeneously so that it can be detected. In particular, ISO26262 demands the use of lockstep execution to avoid CCFs in computation components, whose easiest incarnation is the use of identical cores executing the same task with some staggering (e.g. few cycles of delayed execution for redundant elements) so that their state differs at any point in time and a fault can only lead to different errors in both cores. This is, for instance, the strategy implemented in ASIL-D compliant Infineon AURIX microcontroller units (MCUs) [5]. Fail-safe systems (those with a safe state), therefore, can be built using an ASIL-D MCU and HPC components (e.g. a GPU) so that the latter provides no safety at all, as long as the ASIL-D MCU can safely manage faults in both, the HPC component and the MCU itself.

However, in AD a safe state may not exist anymore since we may not be able to transfer the control back to the hypothetical driver, i.e. the car may even lack a steering wheel. Thus, error recovery capabilities are mandatory for HPC components and thus, we cannot further rely on an ASIL-D MCU to manage the overall safety of the system. Fail-operational systems (those that lack a safe state) impose, therefore, safety requirements even for the HPC components (e.g. ASIL-D compliance).

Manufacturers have already started to sell products targeting AD capabilities, such as the RENESAS R-Car H3 [6] and the NVIDIA Xavier [7] platforms among others. These platforms include multiple general purpose cores (e.g. ARM-based) paired with some accelerators, including a GPU as a key component to process huge amounts of sensed data quickly. So far, those platforms have been regarded as ASIL-B compliant and claimed to be ASIL-C/D capable. However, based on the detailed specifications available, this is only achievable by using redundant functionalities (e.g. based on GPUs and Deep Learning accelerators) [8]. Still, this is a very costly solution since it requires doubling or tripling a significant part of the design and V&V costs, which is against strict cost requirements in the automotive domain. Thus, it becomes mandatory enabling some form of diverse redundancy (i.e. lockstep execution) in the GPU part of the ASIL-C/D functionalities to avoid using fully redundant GPUs. Since in terms of costs and efficiency, setting up two GPUs increases hardware

DOI 10.1109/DSN-S50200.2020.00045

costs and reliability concerns and communications are slower, such lockstep must occur on-chip similarly to the case of the general purpose cores (e.g. Infineon AURIX processors [5]).

In our previous work [9], we have show to what extent diverse redundancy with dual lockstep execution can be obtained by software means in a COTS GPU by launching a redundant kernel for each kernel found in the baseline application. Given that the launching occurs on the CPU (serial) side, the launching itself creates a staggering between the redundant kernels, which adds diversity to the execution by means of timing redundancy. However, dual lockstep execution guarantees error detection, but correction needs full reexecution (reset and restart) or partial reexecution (checkpointing and recovery), even if errors affect a single redundant element (the most common case). Such process may require additional safety measures to tolerate large recovery latencies, such as decreasing driving speed for a short time lapse, which are visible to the user and hence, undesirable. Instead, an alternative providing transparent error recovery for faults affecting a single redundant element are those based on Triple Modular Redundancy (TMR). Those must implement diversity to avoid CCFs (even if reset and restart is needed), but already deliver virtually immediate recover for most faults, which affect a single redundant element.

This paper extends our previous work by moving from dual diverse redundancy in GPUs by software only means to TMR execution with diversity. Our work assesses to what extent commercial off-the-shelf (COTS) GPUs provide already appropriate means to deliver such a solution and identifies gaps that need explicit hardware support.

In particular the contributions of this work are as follows:

- An extension of the software strategy in [9] to implement diverse TMR inside a single COTS GPU.
- Quantitative evidence of how the staggering between redundant execution is created in a single GPU.
- Some discussion on whether those CCFs remaining in our previous work are still present in the case of diverse TMR on COTS GPUs.

II. Background

This section introduces some concepts related to the automotive safety standard ISO26262. In particular, how safety systems are classified into different ASIL, as well as ASIL decomposition. We also provide background on some well-know dependability concepts, but relating them to the needs of safety-critical systems.

A. ASIL in ISO26262

Automotive functionalities inheriting functional safety requirements are classified into different ASIL based on their functional safety risks. The levels go from ASIL-A to ASIL-D, where ASIL-D corresponds to the highest safety risk. Additionally, non-safety-related components are regarded as QM (Quality Managed). The higher the ASIL of an item, the more stringent the safety measures needed to avoid hazardous situations. For instance, error detection (e.g. lockstep execution) and recovery (e.g. reset and restart) features may be required to preserve safety of an ASIL-D MCU. In the context of ISO26262, the automotive safety standard, the ASIL is attached to items based on their safety requirements and a hazard and risk analysis. Such ASIL is then propagated to the different components following some rules (i.e. the default rule consists of validating all components for the same ASIL as the higher level item where they are integrated). However, since increasingly higher ASIL have increasingly higher design and V&V costs, alternative approaches are followed based on what is referred to as *ASIL decomposition*.

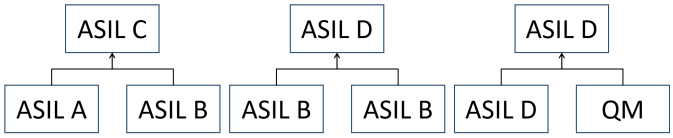

Fig. 1: Examples of ASIL decomposition.

B. ASIL decomposition

Under a given ASIL, some specific diagnostic coverage must be achieved and some random failure rates are deemed as acceptable, being coverage and failure rates more stringent for the highest ASIL. Since reaching certain coverage levels and failure rates may impose excessive cost (e.g. requiring expensive safety measures such as lockstep execution), specific ASIL levels can be reached with the appropriate combination of lower ASIL components. This is illustrated with some examples in Figure 1. For instance ASIL-C can be reached with ASIL-B and ASIL-A items providing *independent redundancy*. Independent redundancy relates to some form of diversity to avoid CCFs, and is explicitly requested in ISO26262.

Such a solution is often used because lower ASIL items are cheaper to design and verify than higher ASIL ones. A common example, is building an ASIL-D MCU by using two ASIL-B cores operating in lockstep. However, in order to apply the *ASIL decomposition*, redundant components must prove to have independent behavior. In the case of computing components, diversity is typically achieved using identical cores and software stacks running with some staggering.

Furthermore, ASIL decomposition is also used for cost reduction trading off availability for fail-safe systems. In particular, a component of a given ASIL (e.g. ASIL D) can be decomposed into, for instance, one ASIL-D component and one or several QM ones, as shown in rightmost example on Figure 1. In this case, the ASIL-D component must be able to preserve safety despite failures of the other components. For instance, as explained before, the HPC part may be deemed as QM, as long as an ASIL-D MCU guarantees error management for errors occurring in the HPC component.

While most systems related to braking and steering resort to some sort of driver intervention to manage potentially hazardous situations, for the highest autonomy levels in AD – levels 3 to 5 as described in J3016 standard [10] – control can only be transferred to the driver in some circumstances (levels 3 and 4) or simply can never be transferred (level 5). Hence, if HPC components need to be included due to the performance constraints in automotive systems and they have to carry tasks related to AD, these accelerators (e.g. GPUs) must be certified to reach ASIL-D on their own. Otherwise, if we combine an ASIL-D low-performance CPU with a QM GPU, being the latter in charge of running ASIL-D processes (e.g. object detection), on a GPU failure, the ASIL-D CPU will be able to

detect the failure, but will not be able to guarantee safety due to the lack of a safe state (i.e. the car must keep taking driving decisions). Therefore, similar solutions to those for MCUs (i.e. diverse redundancy) must be achieved for GPUs and the other accelerators.

C. Independent Redundancy, Diversity and Sphere of Replication

According to ISO26262 software faults and some hardware faults are regarded as systematic, and must be proven that failure risk is residual. Still, random hardware faults cannot be avoided and means are required to prevent them from causing hazards.

Those faults can be caused by, for example, voltage droops, crosstalk, process variations, etc. In order to reach a given ASIL, it must be proven with appropriate diagnostic coverage and failure rate targets that any such single fault cannot lead the system to a hazard. Special care must be taken to protect against CCFs which can affect more than one component at the same time as explained before. For instance, cores implementing DMR could experience a voltage droop and, if no diversity exists, cores could experience identical errors, which would not be detected upon output comparison. Thus, ISO26262 imposes the use of independent (diverse) redundancy for the highest ASIL. As explained before, the most usual solution in automotive systems due to cost reasons is implementing staggered DMR, where the execution in one core is delayed by few cycles w.r.t. the execution in the other core. Staggering requires some buffering to manage inputs and outputs so that staggering is enforced at all times. This is the solution adopted by Infineon AURIX processors for automotive systems [5] as well as some ARM Cortex-R processors [11], [12]. Otherwise, using physically diverse hardware or diverse software impacts noticeably design and V&V costs.

ISO26262 provides no explicit recommendations on how to assess whether diversity has been achieved to a sufficient degree, and quantifying to what extent two different implementations performing the same functionality are diverse is still an open challenge [13]. Hence, diversity is typically assessed qualitatively by safety experts against potential CCFs.

Redundancy can be applied at different granularities according to the *sphere of replication* (SoR). The SoR determines the parts of the system that must be duplicated, how inputs need to be delivered redundantly, and where outputs need to be compared removing redundancy so that redundant components behave externally as if they were a single component. Choosing the right SoR and the number of replicas depends on several tradeoffs like area overheads, re-design costs, fault detection time, and overall system costs. In the context of ISO26262 safety-critical processing components, the SoR is placed at the level of the CPU (core), as done for the AURIX processors. This requires including two replicas of the same core and compare their memory transactions, which requires roughly duplicating computational resources in the chip and being able to ensure that replicas can provide independent behavior. On the other hand, storage (memories, caches) and communication means (buses, crossbars) do not need to be fully replicated and can build upon Error Correction Codes (ECC) and Cyclic Redundancy Check (CRC) as a form of lightweight redundancy with diversity.

As discussed before, DMR and TMR offer different tradeoffs in terms of cost and time to recovery. Obviously, DMR has lower cost, but in the context of automotive, builds on the usual reset and restart recovery actions. However, upon intermittent errors, such recovery solution may not solve the problem in time, thus potentially violating the FTTI of the corresponding functionality. Automotive systems usually perform several retries and, if no recovery is achieved, the system is transferred to a safe state. However, some AD functionalities are fail-operational and no safe state exists. In that case, fault-tolerance may be needed, thus imposing diverse TMR rather than diverse DMR. In our work, we focus on TMR in the context of GPUs, since GPUs are of particular interest of end users and chip vendors as discussed before.

III. Enabling ASIL-D Compliant Fault-Tolerant Operation on COTS GPUs

In this section we introduce the target platform and how safety is preserved in the different components, we analyze the relevant aspects of GPU design and execution model for our work, and present how TMR execution can occur safely on a COTS GPU building on software-only means. Finally, we present a discussion on to what extent TMR and DMR limitations differ.

A. Target Platform

We build our approach on a platform analogous to that introduced in [9], which consists of an ASIL-D compliant MCU together with an HPC accelerator delivering high computation throughput, specifically, a GPU. Such platform is in line with the existing AD platforms. However, the adherence of the GPUs to the requirements of the automotive safety standard, ISO26262, is unknown. Thus, the usage of these accelerators for ASIL-C/D systems requires investigation. Without loss of generality, we focus on NVIDIA GPUs analogous to those in NVIDIA Drive and Xavier automotive families. Still, the findings in this work can easily be extrapolated to other products and manufacturers.

In the proposed platform, the sequential (control) code is executed in the ASIL-D MCU, which deploys lockstep execution, as needed to reach ASIL-D compliance. The ASIL-D MCU offloads intensive (parallel) computations to the GPU, which delivers high computation throughput. These intensive computations are mainly tasks related to AD such as the continuous rendering of the surrounding environment, which involves object recognition and tracking among other high ASIL functionalities. Memory data and on-chip communication during the execution phase of the GPU occur on the same resources (shared) as those used by the ASIL-D MCU and hence, they are naturally protected by specific Error Correcting Codes (ECCs). Additionally, communications between the memory subsystem, the microcontroller and the GPU must be ECC/CRC (Cyclic Redundant Check) protected to guarantee diverse redundancy also on the communication side. A schematic of the proposed hardware platform is showed on Figure 2.

As explained before, such a platform is needed for AD systems and has some specific constraints on the GPU: (1)

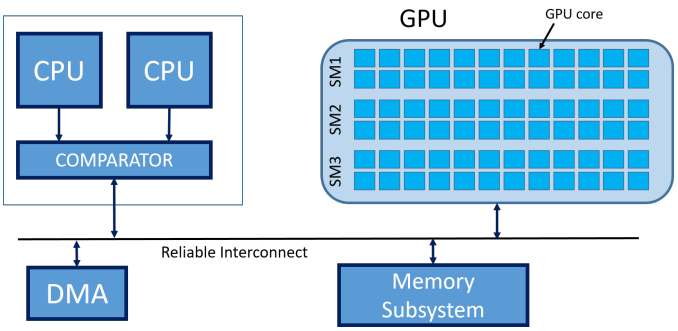

Fig. 2: Proposed computing platform architecture (simplified).

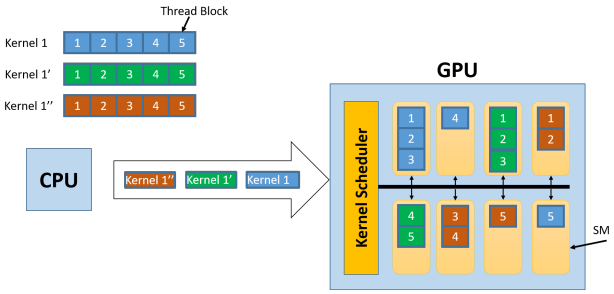

Fig. 3: Spatial and time redundancy in a GPU execution. Three redundant kernels that contain 5 thread blocks each are scheduled in a 8 SM GPU.

it must provide diverse redundancy, (2) TMR is needed to achieve quick error recovery as needed for some time-critical applications, and (3) everything needs to occur within a single GPU for efficiency and reliability reasons.

In this work, we assess to what extent such diverse TMR execution on COTS GPUs can be achieved by software means only to understand whether some hardware support is needed, which would likely be more efficient. Nevertheless, our solution offers the flexibility to be used on COTS GPUs with the aim of avoiding redesigning costs and, potentially, being used on any GPU. For that purpose, we take advantage of the already intrinsic redundant design of the GPUs, whose some relevant aspects for our work we analyze next.

B. Relevant GPU Features

Since different components have different names across GPU vendors, we adhere to NVIDIA nomenclature for the sake of simplicity (and because NVIDIA already targets the automotive domain [7]), but concepts apply to virtually any COTS high-performance GPU.

Figure 2 shows a schematic of the main GPU components relevant for this discussion. GPUs contain multiple Streaming Multiprocessors (SM) (SM_1 to SM_n in the figure). Each one contains the same number of CUDA cores, load/store units and other complex cores. For the sake of the discussion we group them all as simply *cores*. SMs also contain other internal resources such as a shared memory, register files and an internal scheduler among other components. The GPU also contains other resources that are shared across SMs, such as a second level cache (L2), DRAM interfaces and other interfaces, as well as a kernel scheduler.

The CPU dispatches kernels to the GPU, these kernels are dispatched from the kernel scheduler to SMs. Each kernel consists of a number of thread blocks, the kernel scheduler assigns thread blocks to the SMs. When a thread block is assigned to an SM, it is bound to that SM for its entire execution and cannot be migrated, and multiple thread blocks from the same kernel can coexist in the same SM if there are enough resources. However, we assume that at most one kernel can use an SM simultaneously, since removing this constraint increases hardware complexity and performance gains achievable are limited with such fine grain changes. In any case, how thread blocks are scheduled to SMs is an undisclosed feature for the main GPU vendors in general, and NVIDIA in particular.

C. Enabling GPU ASIL-D Operation with Software-only Diverse TMR

To implement our software-only diverse TMR solution on the GPU, we use as SoR granularity the GPU kernel, in line with our previous work [9]. This facilitates generating TMR versions automatically – although we perform such process manually for our evaluation – by replicating inputs and comparing outputs. Note that output variables (as well as input/output variables) need to be replicated to allow for redundant executions and comparisons. Memory transfers between the CPU and the GPU (e.g. *cudaMemCpy* in CUDA) are also replicated.

By launching each redundant kernel using a different CUDA Stream, this allows them to execute in parallel (with some staggering). This is a key point to obtain the physical redundancy, which means ensuring that same computations from the redundant executions are performed in different hardware. As we showed in [9], when multiple kernels are executed using different CUDA Streams at the same time in the GPU, different SMs are allocated to each kernel. This solution splits the computational resources across redundant kernels. Hence, redundant thread blocks are assigned to different SMs as illustrated in Figure 3. Upon an error affecting one redundant execution, fault-tolerance is still preserved. Note that data replication and computation occur in the ASIL-D MCU, thus being already safety compliant. Also, if an error prevents one redundant kernel from finishing, the MCU will detect such circumstance and will provide correct results as long as the other two redundant kernels complete their fault-free execution.

Regarding staggering, which we require to be able to avoid CCFs from affecting more than one redundant execution in the same way, it is achieved automatically due to the serial offloading process on the GPU performed by the CPU. Therefore, kernels are launched one after the other even though they can coexist in the GPU. In particular, when launching a kernel from the CUDA programming model, the CUDA Runtime thread needs to perform several tasks to offload the kernel to the GPU. Later on, in the Evaluation section we deepen on this aspect.

Finally, results are transferred back to the MCU which performs a majority voter algorithm to determine the correct one. Such comparison could also be done faster in the GPU side, but it should be performed in TMR for fault-tolerance

reasons. Thus, for the sake of simplicity and due to the relatively short time to perform such process, we perform it in the MCU side.

D. Diversity Limitations: from DMR to TMR

In our previous work for diverse DMR [9], we identified two main sources of CCFs that cannot be avoided with our software only solution. The first type relates to physical layout effects that may make some specific mask patterns be prone to faults. Since all SMs are, in general, identical, errors induced by layout effects may produce the same error in all redundant copies despite occurring in different SMs at different times. Note, however, that having these effects manifesting for the first time almost simultaneously, and causing *exactly* the same effects in two different physical locations in the chip is unlikely. This relates to effects like process variation, both random and systematic, that may affect different locations differently. In the case of TMR, having three such almost simultaneous first manifestations is even more unlikely to occur. Hence, while this CCF is not avoided completely, TMR is expected to be much less exposed to it than DMR.

The second source of CCFs relates to the use of non-redundant components such as the kernel scheduler, which is unique in GPUs. The serial offloading of the kernels brings some diversity, as well as the fact that the second kernel may find a different scheduler state to that of the first kernel, thus reducing the chances of a fault causing identical errors in both kernels for DMR. In the case of TMR, as for the case of layout effects, the use of 3 redundant copies instead of 2, further decreases the chances of this type of CCF. However, it is not completely avoided and, as in the case of layout effects, hardware support is convenient to guarantee that CCFs are avoided.

IV. Experimental Validation

A. Experimental Setup

We build on Pascal-based NVIDIA GPUs for our evaluation, whose architecture is analogous to that of NVIDIA PX2 AutoChauffer ones used in some cars. The latter are only available to some NVIDIA automotive partners. While not restricted to any particular GPU, we use for our evaluation an NVIDIA GeForce GTX 1080 Ti GPU including 28 SMs with 128 CUDA cores each, and 11GB of GDDR5 memory. On the CPU side, we use an AMD Ryzen 1800X 8-core CPU with 64GB of memory.

Since we lack appropriate AD benchmarks for our evaluation, we use the Rodinia Benchmark Suite [14], [15], often used for GPGPU assessment. Rodinia includes relevant kernels for AD such as those for image processing and pattern recognition.

As explained before, we implement TMR by manually tripling memory allocations, data transfers, and kernel offloading, and performing the output comparison back in the CPU side. Part of our future work involves creating an automatic framework to generate diverse DMR and TMR kernels automatically. Note that no fault injection has been done since TMR is known to tolerate any fault affecting a single redundant instance. Instead, the objective of our evaluation is assessing the execution time impact of implementing software-only diverse TMR in GPUs.

Results are shown in Figure 4. As we can observe, TMR increases execution time w.r.t. DMR, as expected. However, while DMR causes a nearly-linear slowdown w.r.t. the baseline execution time, TMR generally leads to execution times clearly below 3X w.r.t. the non-redundant case. Further investigation reveals that some relatively low contention causes a large impact, and additional contention has a lower impact mostly due to further serialization of the execution.

B. Slack Measurements Results

We have analyzed how serial kernel offloading favors staggering. The sub-procedures that are executed during kernel offloading are: Configure Call, Kernel Setup Arguments and CUDALaunch. We executed 100 times an application from the Rodinia Benchmark Suite (myocyte) modified to use redundant kernel execution and collected measurements using the NVIDIA Profiler (see Figure 5). The line on the top is the slack observed from the start of the first kernel until the start of the second (redundant) kernel, while the stacked bars are the sum of the sub-procedures executed by the CUDA Runtime in the serial CPU when launching the second kernel. The small discrepancy between both values is caused by the NVIDIA profiler, which only provides the execution time of the CUDA calls, neglecting some little CPU code executed in between.

Since each kernel launch on the GPU occurs after the completion of these operations executed on the CPU, and CUDALaunch (the dominant call) has a nearly-constant kernel-independent execution time cost, there will always be a minimum staggering time of few microseconds (10μs in this particular example) across kernel start times, thus enabling implicitly a staggering between redundant kernels. Due to the intrinsics of CPU-GPU interaction, similar behavior is expected for other GPUs based, for instance, on OpenCL.

C. Result Comparison

In order to guarantee that kernel executions on the GPU are correct, a comparison must be done between redundant kernel results in the lockstep MCU. Such comparison could be parallelized and performed (redundantly) on the GPU. Still, our results show that the comparison time is small (less than 1%) for most of the kernels, thus not making worth the effort of porting the comparison to the GPU for most of them.

V. Related Work

ASIL-D compliant MCUs implementing diverse DMR have already been deployed in cars (e.g. Infineon AURIX [5] and the ST Microelectronics SPC56XL70 [16]). However, DMR recovery time may be prohibitive for some fail-operational ASIL-D systems with relatively low FTTI [11]. Some works aim at reducing error detection latencies in these systems by exposing internal MCU contents periodically [17], whereas others aim at reducing recovery latency by means of efficient checkpointing and roll-back [18]. However, AD brings unprecedented performance requirements for safety critical applications requiring safe HPC platforms for a timely and reliable execution [19]. In this context, NVIDIA has recently disclosed a fault-tolerant AD platform building on diverse TMR by running AD software on the GPU, the CPU and an application-specific accelerator [8]. However, such an approach involves different software designs, thus tripling some design and V&V

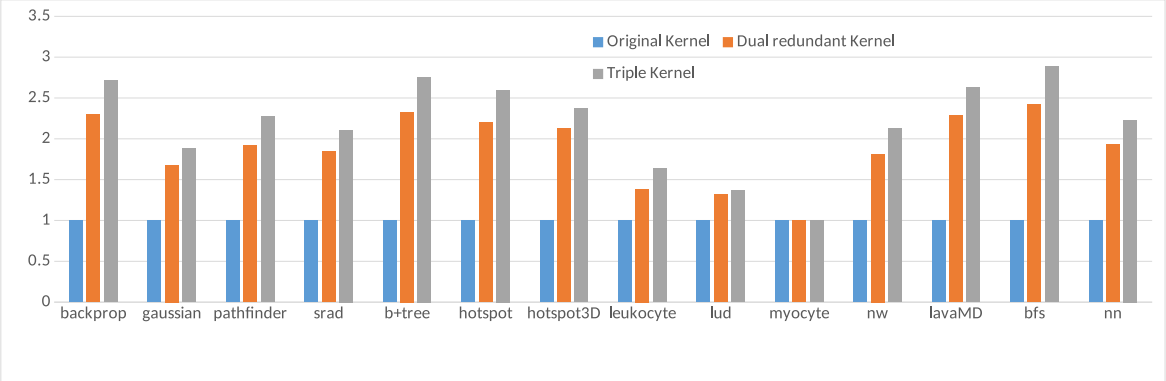

Fig. 4: Execution time of diverse DMR and TMR normalized w.r.t. non-redundant execution.

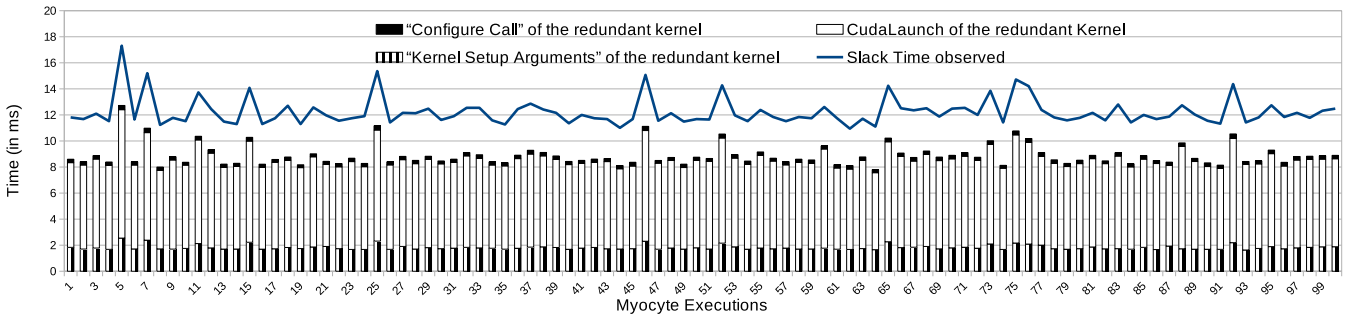

Fig. 5: Slack observed and subprocedures of the kernel launching for the consecutive executions of the Myocyte kernel

costs, which challenges the strict cost constraints in automotive products. Thus, alternative solutions based on hardware and software reuse (e.g. creating diversity by execution staggering) are needed for AD workloads.

NVIDIA labs have released SASSIFI [20], an architecture-level fault injection tool for GPUs based on SASSI [21], which injects instrumentation code at assembly level. However, the GPUs required in order to enable the use of the framework are very limited. Thus, recently SASSI has been deprecated and will be replaced by NVBit [22]. Part of our future work is using NVBit whenever released to perform fault injection campaigns to further verify the effectiveness of our approach.

Some works analyze the different power/performance trade-offs for AD applications when using GPUs, FPGAs or ASIC designs [23]. Their results show that each hardware paradigm is an appropriate fit for at least some AD applications. However, GPUs have the advantage of being already deployed on some commercial AD platforms [2], [6].

Redundancy (without diversity) has been widely studied to deal with random independent faults due to, for instance, radiation, building on time redundancy [24], [25], space redundancy [26], [27] or both of them [28], even for GPUs [29], [30]. However, those works, by not providing diversity, cannot deal with most CCFs. To the best of our knowledge, only the work in [31] provides some hardware support in the kernel scheduler of GPUs to avoid CCFs. Differently to those works, our work targets CCFs, which are of prominent importante in safety-related automotive systems. In particular, our previous work analyzes DMR solutions [9], while this paper considers TMR ones.

VI. Conclusions

This paper analyzes the suitability of COTS GPUs to deliver fault tolerance by implementing software-only diverse TMR, thus extending our previous work on diverse DMR. Our evaluation on a real GPU has shown the following:
(1) Execution times of the kernels were below 3x w.r.t. non-redundant execution in all the cases, proving that the relative costs of redundancy with TMR w.r.t. DMR are lower.
(2) Result comparison has low relative execution time w.r.t. kernel execution, so MCUs can carry out this task.
(3) A minimum initial staggering ($10us$) between the redundant executions is guaranteed by the different subprocedures intrinsic of the CPU-GPU interaction for kernel offloading.

While staggering exists at kernel launching time, part of our future work consists of studying to what extent staggering is preserved during the entire kernel execution. Moreover, we are also very interested on an updated version of SASSIFI, which would allow us to perform fault injection campaigns in the GPU to further validate our approach, and to generate evidence supporting the lower error recovery overheads of TMR w.r.t. DMR for our approach.

Acknowledgements

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 871467 (SELENE). Jaume Abella and Leonidas Kosmidis have been partially supported by the Spanish Ministry of Economy and Competitiveness (MINECO) under a Ramon y Cajal and a Juan de la Cierva Formación postdoctoral fellowship with numbers RYC-2013-14717 and FJCI-2017-34095 respectively.

REFERENCES

[1] SC Lin et al., "The architectural implications of autonomous driving: Constraints and acceleration," in *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '18. New York, NY, USA: ACM, 2018, pp. 751–766. [Online]. Available: http://doi.acm.org/10.1145/3173162.3173191

[2] TESLA, "Full Self-Driving Hardware on All Cars," https://www.tesla.com/autopilot.

[3] International Standards Organization, *ISO/DIS 26262. Road Vehicles – Functional Safety*, 2009.

[4] J. Espinosa et al., "Analysis and RTL Correlation of Instruction Set Simulators for Automotive Microcontroller Robustness Verification," in *DAC*, 2015.

[5] Infineon, "AURIX Multicore 32-bit Microcontroller Family to Meet Safety and Powertrain Requirements of Upcoming Vehicle Generations," 2012, http://www.infineon.com/cms/en/about-infineon/press/press-releases/2012/INFATV201205-040.html.

[6] "RENESAS R-Car H3," https://www.renesas.com/en-us/solutions/automotive/products/rcar-h3.html.

[7] D. Shapiro, "Introducing Xavier, the NVIDIA AI Supercomputer for the Future of Autonomous Transportation," *NVIDIA blog*, 2016. [Online]. Available: https://blogs.nvidia.com/blog/2016/09/28/xavier/

[8] NVIDIA, "NVIDIA Announces World's First Functionally Safe AI Self-Driving Platform," 2018, https://nvidianews.nvidia.com/news/nvidia-announces-worlds-first-functionally-safe-ai-self-driving-platform.

[9] S. Alcaide et al., "Software-only diverse redundancy on gpus for autonomous driving platforms," in *2019 IEEE 25th International Symposium on On-Line Testing and Robust System Design (IOLTS)*, July 2019, pp. 90–96.

[10] SAE International, *J3016: Taxonomy and Definitions for Terms Related to On-Road Motor Vehicle Automated Driving Systems*, 2014.

[11] X. Iturbe et al., "Addressing Functional Safety Challenges in Autonomous Vehicles with the Arm Triple Core Lock-Step (TCLS) Architecture," *IEEE Design and Test*, vol. 35, no. 3, pp. 1–1, 2018.

[12] B. Venu et al., "A Fail-Functional Automotive CPU Subsystem Architecture for Mitigating Single Point of Failures," in *IEEE International Workshop on Automotive Reliability and Test*, 2017.

[13] S. Alcaide et al., "DIMP: A low-Cost Diversity Metric based on circuit Path analysis," in *DAC*, 2017.

[14] S. Che et al., "Rodinia: A benchmark suite for heterogeneous computing," in *IISWC*, 2009.

[15] ——, "A characterization of the Rodinia benchmark suite with comparison to contemporary CMP workloads," *IISWC*, 2010.

[16] STMicroelectronics, "32-bit Power Architecture microcontroller for automotive SIL3/ASILD chassis and safety applications," 2014.

[17] C. Hernandez et al., "Timely Error Detection for Effective Recovery in Light-Lockstep Automotive Systems," *IEEE TCAD*, vol. 34, no. 11, 2015.

[18] ——, "Low-cost checkpointing in automotive safety-relevant systems," in *DATE*, 2015.

[19] ARM, "ARM Expects Vehicle Compute Performance to Increase 100x in Next Decade," 2015, https://www.arm.com/about/newsroom/arm-expects-vehicle-compute-performance-to-increase-100x-in-next-decade.php.

[20] SKS Hari et al., "Sassifi: An architecture-level fault injection tool for gpu application resilience evaluation," in *2017 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, April 2017.

[21] M. Stephenson et al., "Flexible software profiling of gpu architectures," in *2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA)*, June 2015.

[22] O. Villa et al., "Nvbit: A dynamic binary instrumentation framework for nvidia gpus," in *Proceedings of the 52Nd Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '52, 2019. [Online]. Available: http://doi.acm.org/10.1145/3352460.3358307

[23] S. Lin et al., "The architectural implications of autonomous driving: Constraints and acceleration," in *ASPLOS*, 2018.

[24] A. Mahmoud et al., "Optimizing software-directed instruction replication for gpu error detection," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis*, 2018. [Online]. Available: https://doi-org.recursos.biblioteca.upc.edu/10.1109/SC.2018.00070

[25] M. B. Sullivan et al., "Swapcodes: Error codes for hardware-software cooperative gpu pipeline error detection," in *MICRO*, 2018.

[26] D. A. G. Oliveira et al., "Modern gpus radiation sensitivity evaluation and mitigation through duplication with comparison," *IEEE Transactions on Nuclear Science*, vol. 61, no. 6, 2014.

[27] M. Dimitrov et al., "Understanding software approaches for gpgpu reliability," in *GPGPU*, 2009.

[28] J. Wadden et al., "Real-world design and evaluation of compiler-managed gpu redundant multithreading," in *ISCA*, 2014.

[29] C. Kalra et al., "Performance evaluation of compiler-based software rmt in an hsa environment," 03 2016.

[30] M. Gupta et al., "Compiler techniques to reduce the synchronization overhead of gpu redundant multithreading," in *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*, June 2017, pp. 1–6.

[31] S. Alcaide et al., "High-integrity gpu designs for critical real-time automotive systems," in *2019 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2019.