Safety-Related Challenges and Opportunities for GPUs in the Automotive Domain

GPUs have been shown to cover the computing performance needs of autonomous driving (AD) systems. However, since the GPUs used for AD build on designs for the mainstream market, they may lack fundamental properties for correct operation under automotive's safety regulations. In this paper, we analyze some of the main challenges in hardware and software design to embrace GPUs as the reference computing solution for AD, with emphasis in ISO 26262 functional safety requirements.

Sergi Alcaide
Universitat Politècnica de Catalunya
Barcelona Supercomputing Center (BSC)

Leonidas Kosmidis
Barcelona Supercomputing Center (BSC)

Hamid Tabani
Barcelona Supercomputing Center (BSC)

Carles Hernandez
Barcelona Supercomputing Center (BSC)

Jaume Abella
Barcelona Supercomputing Center (BSC)

Francisco J. Cazorla
Barcelona Supercomputing Center (BSC)

IIIA-CSIC

INTRODUCTION

High-performance embedded systems are increasingly used in critical domains such as transportation (road vehicles, airplanes, trains), industrial machinery, health devices and satellites. Engineers of Critical Real-Time Embedded Systems (CRTES) develop products following commonly accepted best practices and, in the case of safety-critical systems, showing compliance with legal directives is mandatory before they are allowed to operate. This requires going through a certification process as defined by applicable safety standards, e.g., ISO 26262 [1] for road vehicles.

The increasing use of 'smart' software functionalities as the main competitive factor in CRTES is relentless. In automotive, the main software functionality relates to driving automation, whose potential benefits range from the reduction of accidents and the CO_2 footprint to increasing people's quality of life by reducing the time they spend driving. These benefits have boosted the trend towards full automation with most mid- and high-end cars already featuring some Advanced Driver Assistance Systems (ADAS) and the first Autonomous Driving (AD) Level 3 car already in mass production (the Audi A8). Note that there are five Autonomous Driving Levels from 0, or no automation, to level 5: fully automation in which driving is automatically handled in (all) scenarios as complex as those as human drivers can encounter on the roads.

AD functionality can be broadly classified into the perception of the environment surrounding the vehicle, localization to estimate vehicle's position, planning the vehicle trajectory, and controlling vehicle actuators. Perception, the most compute intensive module, builds on object detection and tracking. In the past few years, impressive improvements in machine learning (ML) techniques, e.g. Deep Neural Networks (DNNs), have dramatically changed state-of-the-art algorithms for perception by achieving significantly higher accuracy. This has made ML-based perception techniques the preferred solution in industry. The other side of the coin is that ML techniques carry unprecedented performance demands in automotive. The performance requirements of ADAS alone are projected to increase by 100x from 2016 to 2024 [2].

Initial research studies [3] and performance data from chip vendors show the effectiveness of GPUs to accelerate ML-based libraries for AD. This has attracted the attention of car manufacturers who have started analyzing GPU's potential to cover AD's performance requirements.

Since high-end GPUs targeting AD systems build on designs for the mainstream market, they may find some difficulties to adapt to automotive's specific requirements. In this paper, we analyze some of the main challenges that GPUs, and the software running on them, will face providing safety assurance in accordance with ISO 26262 functional-safety standard. We also cover other relevant challenges such as time predictability. Specifically, the main contributions of this work are:

1. At the hardware level, harsh operation conditions (e.g., extreme temperatures) makes GPUs more vulnerable to random hardware faults. Resorting to ISO 26262 standard solutions such as diversity and redundancy has to be done cautiously in GPUs. For instance, while GPUs naturally offer redundancy sources, they must be carefully exploited to preserve high performance and prevent a single fault from becoming a common cause failure in all redundant instances, which could lead the system to a hazardous situation.

2. At the software level, we identify these challenges.

- The AD builds on generic, i.e., non-automotive specific, ML libraries. This allows car makers to enjoy the improved functionality (e.g., higher object detection accuracy) of the latest available generic ML libraries, which see a new release every few months. However, their *generality* increases the probability that the libraries implement hard-to-validate features, increasing the effort to assess libraries' adherence to ISO 26262 guidelines on software coding and development. As an illustrative example, our results with YOLO v3, a state-of-the-art object detector system, show that the code coverage achieved, a basic software unit structural coverage metric, is well below the 100% needed in ISO 26262.
- For performance-improving reasons, ML libraries build on low-level GPU-optimized libraries such as `cuDNN`. The black-box, i.e., closed-source, nature of these libraries, however, challenges assessing their adherence to ISO 26262 guidelines for software. This requires library owners to undergo the certification process or the use of open-source libraries that provide similar performance to their closed-source counterparts, so that end users take care of its certification. In either the case, changes to simplify validation and verification can reduce efficiency and result in the creation of ISO 26262-specific branches of the libraries.
- Languages to program GPUs make use of features that hamper software (code) verification activities that already amount for most of the total development effort for the highest safety levels (ASIL D). As illustrative examples, we discuss two well-known features: the use of pointers (that must be prevented) and the use of defensive programming to prevent systematic software faults (that must be favored). We show that ISO 26262-aware programming languages prevent (favor) some of those undesired (desired) features while maintaining GPU's performance benefits.

3. Time predictability,[4] a fundamental requirement of CRTES, is another relevant challenge in GPUs since it is hard to achieve on (complex) high-performance hardware like GPUs. To address this challenge, we advocate for hardware support to increase *observability* as a necessary element to obtain evidence of the correctness of the derived Worst-Case Execution Time (WCET) estimates. Further, it is required to analyze those elements in DNN codes negatively affecting predictability, which have only been superficially explored so far.

Overall, while adherence to ISO 26262, already achieved for ADAS, is challenged by AD. Under ADAS, the computing system acts as a fail-safe system and in case of misbehavior, it returns the control to the driver. However, AD makes some systems fail-operational preventing the control to be returned to the driver. This has onerous consequences on the safety solutions adopted to guarantee that the system remains operational upon a fault (aka fault-tolerant).

Recently, the NVIDIA Xavier has been announced as an ISO 26262-capable GPU-based SoC for AD. While detailed technical specifications on how this SoC achieves fail-operational capabilities are not yet available, achieving ISO 26262 compliance requires appropriate redundancy and diversity strategies. To our understanding, in the NVIDIA platform, this is achieved with a non-negligible amount of replication of functionalities (e.g. using GPUs and Deep Learning Accelerators), which may significantly increase V&V costs due to the use of two different software and hardware implementations, or may lead to inefficient solutions since execution time will be

dominated by the slowest implementation. In general, it remains unclear to what extent AD systems can be efficiently deployed and validated in a cost-effective manner on an AD-capable SoC. In this paper, we analyze some of the most relevant challenges related to this matter.

BACKGROUND ON SAFETY AND AD SOFTWARE

Hardware accelerators can provide the computing performance required to execute in real-time artificial intelligence applications, making them central elements to enable AD in automotive. GPUs, in particular, are already on the roadmap of many car manufactures to cover the performance needs of AD. Yet, several fundamental questions remain unanswered on how GPUs will cover other functional and non-functional requirements of safety-critical systems.

The automotive functional safety standard ISO 26262, defines 4 Automotive Safety Integrity Levels (ASIL) varying from ASIL-A (lowest criticality) to ASIL-D (highest criticality). Besides, the Quality Management (QM) category covers those components that cannot cause safety risks upon a failure.

Certification requires evidence that the risk of systematic failures is residual and that failures due to random hardware faults do not exceed specific probability bounds. In general, the higher the criticality, the more evidence required to show compliance with safety requirements, and the lower the probability bounds. Also, whether the system is *fail-safe* (i.e., the system can reach a safe state upon a failure) or *fail-operational* (i.e., the system must remain operational, independently of the presence of a failure) plays a key role defining the mitigation techniques needed to meet those failure probability bounds.

ASIL allocation. Functional safety conformance requires identifying the safety goals of the system. Those goals are subsequently mapped into safety requirements that are propagated to components as the system is decomposed until reaching atomic software and hardware components, which are finally implemented and integrated. The higher the severity and exposure to hazards, and the lower the controllability, the higher the ASIL allocated to the component. As an example, in general, most systems related to steering and braking are ASIL D, and hence, AD operation must also adhere to ASIL D certification.

Redundancy and diversity. Under ISO 26262 ASIL C/D – as required by AD systems – can be achieved by means of redundant and diverse implementations. This requires setting up redundant instances of an item such that they are not subject to common cause failures. Diversity can be achieved in multiple ways. For instance, one may set up different systems performing the same functionality (e.g., one based on lidar and another based on a camera), or the very same system may be replicated introducing diversity at lower levels (e.g., different software implementations, staggered execution).

SAFETY ASSURANCE: IMPACT ON HARDWARE

A. *ASIL Decomposition*

Traditionally, automotive systems have been considered fail-safe, which requires simpler measures to guarantee adherence to ISO 26262, such as returning the control to the driver. However, the transition to fail-operational systems driven by level-5 AD significantly complicates achieving functional safety and hampers some forms of ASIL decomposition that were traditionally employed to save development costs.

ASIL decomposition is used (i) to implement high-ASIL components with redundant and sufficiently independent lower-ASIL components (see top example in Figure 1 in which two ASIL B components can be used to reach ASIL D) and (ii) to allow a subset of the components preserve safety, thus remaining at the corresponding ASIL level, whereas others are regarded as Quality Managed (QM) since, on a failure, ASIL components will detect it and keep the system safe. Such decomposition is often used to keep monitoring functionalities at the corresponding ASIL level (e.g., ASIL D in the bottom example in Figure 1), whereas computation components are regarded as QM. On a failure of the QM component, the monitoring one detects it and moves the system to a safe state, thus impacting availability but not safety. Whether faults occur often is, therefore, a matter of availability and so, business, but functional safety is preserved.

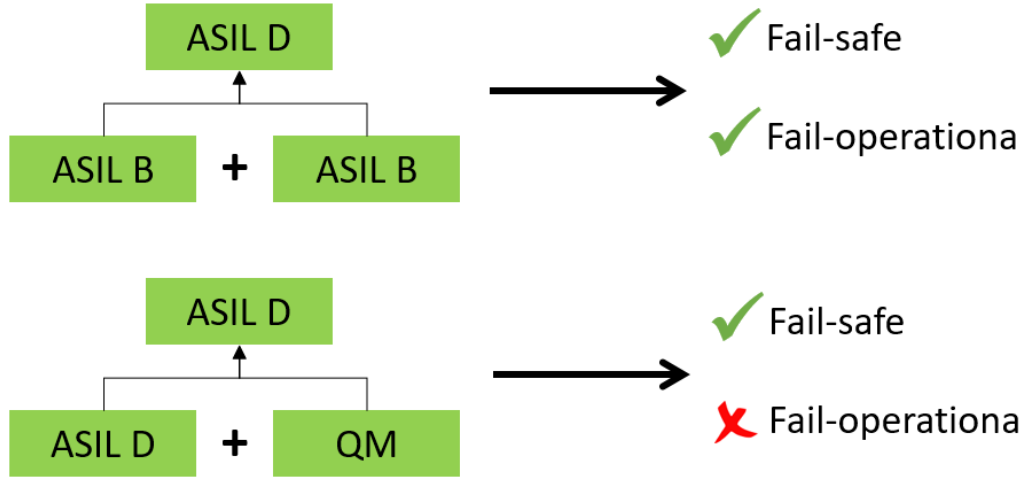

Figure 1. Examples of ASIL decomposition and appropriateness for fail-safe and fail-operational systems.

In AD, since some ASIL C/D functionalities are fail-operational (e.g., those related to braking and steering), the components implementing such functionality must achieve the corresponding ASIL, and ASIL decomposition cannot be applied to keep those items as QM since a safe state may not exist at all. Hence, some form of fault tolerance must be incorporated to keep the system operational despite faults.

B. Redundancy and Diversity

GPUs naturally offer lots of hardware redundancy that can be exploited to implement diversity solutions. However, for IP confidentiality reasons, some GPU's internal behaviors (e.g., resource allocation) are managed automatically by hardware, i.e., in a black box manner. Unfortunately, this practice clashes with guaranteeing diversity, since low-level management of the resources from software may be needed.

Yet, it is in our view that those issues are not roadblocks for the use of GPUs for AD in the automotive domain. That is, the type of homogeneous hardware redundancy offered by GPUs can be made compatible with automotive needs, similar to the case for homogeneous cores operating in lockstep mode. For instance, identical cores, despite being homogeneous in terms of front-end design, provide diversity by several means like operating with some time shift so that activities carried out at any given time differ across cores and hence, upon a fault affecting both cores, the effect is necessarily different and thus, errors can be detected timely. Another diversity technique usually employed in lockstep cores is the use of layout diversity. Similar approaches can be enabled on top of GPUs as long as common cause failures are avoided by construction by, for instance, using similar concepts as for cores (e.g. allocating separate sets of resources to each redundant thread and operating with some time shift). Also, GPU architectures may evolve and match ISO 26262 requirements in the future since modifications will likely have a roughly negligible impact in cost and performance, and GPU vendors like NVIDIA already acknowledge the need for ISO 26262 compliance [5].

Example: A typical example of redundancy in the context of ISO 26262 is lockstep execution which, in the case of Infineon AURIX processors (e.g., TC29x processor family) is performed with staggered execution so that a given fault does not cause the same faulty output on both instances. However, regular lockstep designs cannot provide fail-operational levels. For instance, regular lockstep systems (referred to as 1oo2, meaning one out of two) rely on the fact that, on a fault, it will be detected by comparing the output of both items, which will differ. In fact, such output can be faulty for both items but, as long as it differs due to some source of diversity (e.g., staggered operation), it guarantees fault detection. Unfortunately, fail-operational systems can only build upon 1oo2 implementations when fault detection is guaranteed and either (1) one of the items is necessarily non-faulty, and the non-faulty item can be identified so that regular operation can be preserved; or (2) on a fault, the functionality can be re-executed correctly before causing a

hazard, which however, is a complex process. Hence, reaching efficient ASIL C/D for fail-operational systems remains as an open challenge for the automotive industry, which might require 2oo3 redundancy, plus some form of voting scheme to keep the system fault-tolerant.

Summary: GPUs massive parallelism allows supporting NooM (where N < M) redundancy. However, two key open challenges remain. First, guaranteeing diversity by avoiding common cause failures. And second, excessive use of 1oo2 (or 2oo3) may result in unaffordable procurement and energy costs, which calls for providing efficient NooM redundancy solutions.

C. *Ability to Operate in Harsh Environments*

Hardware qualified for automotive use needs to have an operating temperature that ranges between -40 ºC and 150 ºC for the highest criticality (Grade 0 Automotive Electronics) and increased reliability requirements for soft errors. While those operation conditions are much more challenging than those for server or office electronic equipment, they are affordable for GPUs by employing appropriate circuit designs such as larger transistors and wider wires. However, those design practices may cause some performance degradation due to the use of slower circuits, which may also consume more power. Still, in our view, suitable tradeoffs can be found.

SAFETY ASSURANCE: IMPACT ON SOFTWARE

A. *Coding Standard and Architectural Design*

Critical software across all sectors needs to comply with coding and development guidelines, in order to facilitate its validation and certification against the standards of the particular domain. In automotive, ISO 26262, for instance, recommends the limited use of certain features that complicate the certification of software applications such as pointers and dynamic memory allocation. It also encourages the use of safe language subsets to limit the use of error-prone language features. For instance, MISRA C is a subset of the C language that defines a set of rules that can be statically checked by commercial tools and therefore enforce their use. Besides coding standards, ISO 26262 defines (i) requirements on the architecture of critical software that must exhibit properties such as modularity, encapsulation and minimal complexity; (ii) verification methods of the safety requirements including source code review (walk-through and inspection) and source code analysis (control flow, data flow, static code); and testing methods – used to verify software and ascertain its quality. For instance, at the unit testing level, ISO 26262 requires structural code coverage such as statement and branch coverage.

GPU software is based on low-end C-like APIs like CUDA and OpenCL, which bring some challenges to show adherence to ISO 26262. These challenges include the following:

Use of Pointers. CUDA and OpenCL programs use pointers as an indispensable feature of their programming model, since the programmer has to explicitly allocate and maintain two separate sets of pointers, one for the host memory and one for the device memory. Moreover, the programmer also has the responsibility to perform memory transfers between these two memory spaces. Note that recent versions of both CUDA (6 and later) and OpenCL (2.0) provide two equivalent features called Unified Virtual Memory and Shared Virtual Memory respectively. These features simplify the programmability and enhance productivity by taking care of the transfers implicitly, providing the user with the abstraction of a single address space. However, they might incur a performance penalty while introducing another black-box in the timing and functional behavior, which complicates the certification of the system as we discuss in the following Section. Moreover, even with these features, pointers are still present in the programing model.

Brook is a stream-programming language targeting GPUs. In the same way, MISRA C constraints C, Brook Auto [6] defines a subset of Brook rules that are certification friendly, without limiting the expressiveness of the language. For instance, Brook Auto does not expose pointers to the programmer, and takes care of those tasks automatically, reducing the possibility of human errors.

Example: Figure 2 (left) shows an illustrative example of Brook Auto that highlights some of its benefits. The sample program launches a GPU kernel that operates on two input data vectors (*streams* in Brook terminology) and generates its result in a third data vector. In the program that calls the kernel, there are two versions of each vector required, one for the host (suffixed " h") and one for the device (suffixed " d"), shown in the lines 14 and 15 respectively. The same code written in CUDA, see Figure 2 (right), shows that pointers are required both in the GPU, for passing data

in the kernel (line 1), as well as on the host side for allocating memory (lines 14-16) and manage the transfers between host and GPU buffers (lines 18-19 and 21). Note that the OpenCL version of the code has the same characteristics as CUDA, but it is more verbose. Therefore it is omitted for clarity. Brook uses statically defined streams that prevent explicit memory allocations (*cudaMalloc*) and low-level memory management, which could result in programming mistakes due to wrongly supplied size parameters or memory exhaustion. Streams cannot be directly accessed (e.g., indexed) from the host side, since this would result in a compilation error and they can only be accessed using certain API calls (*streamRead* and *streamWrite*) to copy data from and to host buffers. Stream size is integrated within these calls, preventing out of bounds accesses from the host side.

```
1
2  #define MAX_ITERS 10000
3
4  kernel void foo(float a<>, float b[], out c<>){
5    float acc=0.0;
6    for(int i<0; i < a || i < MAX_ITERS; i++){
7      acc +=  b[indexof(c).x];
8    }
9
10   c = a + acc;
11 }
12
13 int main(void){
14   float a_h[100], b_d[100], c_h[100];
15   float a_d<100>, b_d<100>, c_d<100>;
16
17   streamRead (a_d, a_h);
18   streamRead (b_d, b_h);
19   foo (a_d, b_d, c_d);
20   streamWrite (c_d, c_h);
21 }
22
```

```
1  __global__ void  foo(float * a, float * b, float * c){
2    unsigned int tid = blockIdx.x*blockDim.x + threadIdx.x;
3    float acc=0.0;
4    for(int i < 0; i < a[tid]; i++)
5      acc += b[tid];
6
7    c[tid] = a[tid] + acc;
8  }
9
10 int main(void){
11   float a_h[100], b_d[100], c_h[100];
12   float * a_d, * b_d, * c_d;
13
14   cudaMalloc(&a_d, 100*sizeof(float));
15   cudaMalloc(&b_d, 100*sizeof(float));
16   cudaMalloc(&c_d, 100*sizeof(float));
17
18   cudaMemcpy(a_d, a_h, 100*sizeof(float), cudaMemcpyHostToDevice);
19   cudaMemcpy(b_d, b_h, 100*sizeof(float), cudaMemcpyHostToDevice);
20   foo<<<1,100>>>(a_d, b_d, c_d);
21   cudaMemcpy(c_h, c_d, 100*sizeof(float), cudaMemcpyDeviceToHost);
22 }
```

Figure 2. Same code programmed with Brook Auto (left) and CUDA (right)

From the kernel side, streams can be accessed in two ways. Regular streams in which each GPU thread accesses its corresponding element in the array, declared as '<>' and *gather* streams, which are declared with '[]'. In the former case, Brook Auto takes care of accessing the correct element, while in the latter it suppresses potentially illegal out of bound accesses ensuring fault isolation.

Other Dynamic Features. Brook Auto also restricts dynamic language features that can lead to deadlocks or complicate the WCET analysis of the software. For example, notice that the kernel in the Brook Auto example contains an extra defensive-programming condition in the loop (line 6). This condition restricts the number of iterations of the kernel to a statically defined upper bound limit, although the main loop condition is input dependent. The absence of such a statically computed condition would result in a compilation failure, thus enforcing this rule. On the contrary, the CUDA version is unprotected of this type of programming risks, which complicate GPU software certification with ISO 26262.

B. *Generic ML and Black-Box CUDA Libraries*

Generic ML Libraries. The dramatic increase in ML usage in a variety of domains makes that leading artificial intelligence companies provide several widely-used frameworks and highly optimized libraries to facilitate and make better use of available platforms and architectures [7]. Like in many other areas, state-of-the-art AD systems strongly rely on these libraries and use them extensively. However, not only the algorithms but also these frameworks and libraries are quite generic, designed based on totally different objectives to those of CRTES in general and AD particular, which challenges providing evidence that software achieves its safety requirements. In fact, to our knowledge, no study has been carried out on the adherence of those libraries to ISO 26262 software requirements.

Example. As an illustrative example, we focus on statement coverage, a basic structural coverage metric at the software unit level. In particular, we run the YOLO v3, which is a state-of-the-art object detector widely used in real AD systems comprising more than 20 functions. We run several real-scenario tests and measure simple statement and branch coverage using RapiCover low-overhead coverage analysis tool [8]. The former captures the fraction of static instructions (those in the binary) executed in the tests, and the latter the fraction of program branches or conditional states triggered during the tests. Obtained results are shown in Figure 3.

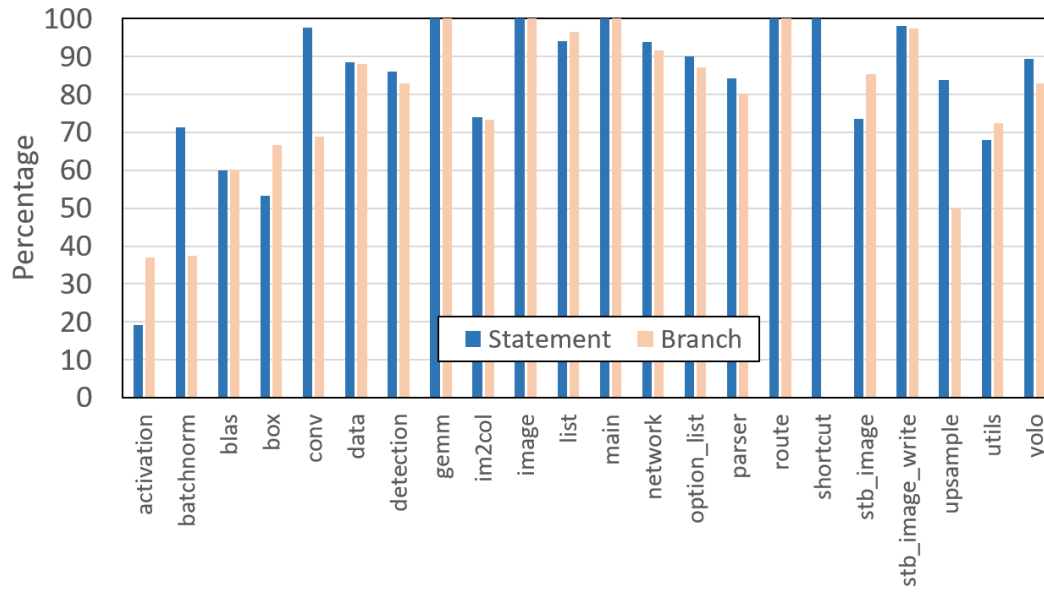

Figure 3. Code Coverage for YOLO v3

Each column represents all the functions in each file. Note that, despite excluding all YOLO functions that were not called, both branch and statement coverage are very low. Average coverage is 83% and 79% for statement and branch respectively, and as low as 19% and 37% respectively for individual files. While ISO 26262 does not specify coverage targets, its parent standard, IEC61508 (Functional Safety of Electrical/Electronic/Programmable Electronic Safety-Related Systems) recommends 100% coverage for all metrics. Hence, the coverage levels observed for YOLO are not acceptable for any ASIL since either branch or code statement are *highly recommended* ('++') for all ASIL levels.

It is also worth noting that the concept of code coverage has not even been defined for GPU code. The fact that GPU instructions are SIMT (Single Instruction, Multiple Thread) and warp divergence (predicated execution) complicate simply extending CPU code coverage to GPUs.

Low-level CUDA Libraries. Libraries used for artificial intelligence and machine learning in AD – as the majority of widely-used operations in GPUs – rely on highly optimized closed-source libraries (e.g., `cuBLAS` and `cuDNN`). From a functional safety point of view, these are black-boxes without detailed information on their implementation, code, and algorithm. This might prevent their safety analysis by end users, e.g., source code analysis and code coverage, a mandatory requirement of ISO 26262. In our view, overcoming this limitation requires one of the following:

- The use of open-source libraries, which must provide competitive performance. For instance, results show that CUTLASS,[9] NVIDIA's open-source collection of CUDA C++ templates and abstractions for implementing high-performance GEMM computations, provides very close performance in comparison with cuBLAS.
- Closed-source libraries owners go through the certification process and adapt their libraries to fit ISO 26262 requirements.

Code changes to achieve ISO 26262 adherence can, however, cause performance loss with respect to the original performance-improving centric code, which is not acceptable in other non-critical domains in which these libraries are used. This can result in the creation of branches of the code specific for the automotive domain, with increased development and maintainability costs.

C. *Domain Specific Optimizations*

Numerous schemes have been proposed to optimize deep learning models (e.g., layer removal and fusion). From those, calibrating the neural network models for lower precision (a.k.a quantization) is one of the commonly-used schemes. In general, these optimizations aim at delivering lower latency and higher throughput for deep learning inference applications and/or reduce the energy profile. However, some of these optimizations can come at the expense of increasing the probability of producing a wrong result by the application. Hence, these approaches directly affect the

accuracy of the application within a considerable and wide margin depending on the input data. For a critical domain such as automotive, these schemes decrease the decisiveness of the application. Hence such optimizations must be used with caution in AD factoring their impact on overall's application accuracy.

D. *Time Predictability*

In CRTES, functionalities need to be completed within certain timing bounds, called deadlines. Hardware and software architectures in CRTES require time predictable timing behavior that allows deriving tight and reliable WCET estimates [10]. WCET analysis of GPU software is still in a very early stage [11]. Static timing analysis has been performed under very limited scenarios, such as assuming that the kernel is executed on a single streaming multiprocessor, while measurement-based analysis on the other side has been performed without providing enough evidence that the worst-case scenarios have been exercised. Both solutions are negatively affected by the existence of many undocumented features in GPU architecture and software, contributing to analyzing their real-time properties analysis hard, compared to the CPU architectures used traditionally. In our view, a way to alleviate this problem is by increasing GPU observability. In particular, a more powerful set of WCET-aware monitors (performance monitoring counters) helps to provide insightful information on application worst-case behavior when run on the target hardware as an instrumental element to build a safety argument [12]. On the timing analysis side, the use of statistical-based approaches is on the rise as it fits the increasing execution-time variability applications suffer when running on complex processors such as GPUs [3-4].

CONCLUSION

As the software component to implement safety-related functionality continues to increase in cars, so do its performance requirements and the guarantees required on its correct behavior. The former is covered in a cost-effective manner by deploying software and hardware-accelerator techniques originally designed for other high-performance, i.e., noncritical, domains. As we have discussed in this paper, the sustainability of this approach builds on developing well-designed adaptations to address key challenges when satisfying safety regulatory standards. The overall ISO 26262 philosophy builds on defining a set of requirements and a set of tests, which emanate from the requirements, that are used to assess whether a particular software implementation is correct. Whether this approach can be directly applicable to ML-based code is still an open question due to the difficulties in defining whether, for instance object detection software works properly, and defining the tests to assess so. Overall, new interpretations and analysis of how to certify software with respect to that in place by ISO 26262, might be necessary.

ACKNOWLEDGMENTS

Authors would like to thank Guillem Bernat from Rapita Systems for his technical feedback on this work. The research leading to this work has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 772773). This work has also been partially supported by the Spanish Ministry of Science and Innovation under grant TIN2015-65316-P and the HiPEAC Network of Excellence. Jaume Abella has been partially supported by the Ministry of Economy and Competitiveness under Ramon y Cajal postdoctoral fellowship number RYC-2013-14717. Carles Hernández is jointly funded by the Spanish Ministry of Economy and Competitiveness and FEDER funds through grant TIN2014-60404-JIN.

REFERENCES

1. *ISO/DIS 26262. Road Vehicles - Functional Safety*, ISO Standard 43.040.10, Nov. 2011.
2. ARM, "ARM expects vehicle compute performance to increase 100x in next decade," ARM, Cambridge, U.K., Apr. 23, 2015. [Online]. Available: https://www.arm.com/about/ newsroom/arm-expects-vehicle-compute-performance-to-increase-100x-in-next-decade. php.

3. S.-C. Lin *et al*., “The architectural implications of autonomous driving: Constraints and acceleration,” in *Proc. 23rd Int. Conf. Archit. Support Program. Lang. Oper. Syst.*, 2018, PP. 751–766.
4. F. J. Cazorla *et al*., “Reconciling time predictability and performance in future computing systems,” IEEE Des. Test, vol. 35, no. 2, pp. 48–56, Apr. 2018.
5. J. Zander, “Functional safety for autonomous driving,” in *Proc. Auton. Veh. Mach. Conf.*, 2017.
6. M. M. Trompouki and L. Kosmidis, “Brook Auto: High-level certification-friendly programming for GPU-powered automotive systems,” in *Proc. 55th Annu. Des. Autom. Conf.*, 2018, pp. 1–6.
7. M. Abadi et al., “Tensorflow: A system for large-scale machine learning,” in *Proc. 12th USENIX Conf. Oper. Syst*. Des. Implement., 2016, pp. 265–283.
8. Rapita Sytems Ltd., “RapiCover.” [Online]. Available: https://www.rapitasystems.com/ products/rapicover.
9. A. Kerr *et al*., “CUTLASS: Fast linear algebra in CUDA Cþþ,” blog. [Online]. Available: https://devblogs.nvidia.com/cutlass-linear-algebra-cuda/.
10. R. Wilhelm *et al.*, “The worst-case execution-time problem—Overview of methods and survey of tools,” *ACM Trans. Embedded Comput. Syst*., vol. 7, 2008, article 36, pp. 1–53.
11. A. Betts and A. Donaldson, “Estimating the WCET of GPU-accelerated applications using hybrid analysis,” in *Proc. 25th Euromicro Conf. Real-Time Syst.*, 2013, pp. 193–202.
12. E. Mezzeti *et al*., “High-integrity performance monitoring units in automotive chips for reliable timing V&V,” *IEEE Micro*, vol. 38, no. 1, pp. 56–65, Jan./Feb. 2018.

ABOUT THE AUTHORS

Sergi Alcaide is a PhD student in Universitat Politècnica de Catalunya (UPC) and Barcelona Supercomputing Center (BSC). His research interests include computer architecture and reliability. He obtained his Master in Innovation and Research in Informatics at UPC. Contact him at salcaide@bsc.es.

Leonidas Kosmidis is a senior researcher in BSC. He has a PhD in computer architecture from the Polytechnic University of Catalonia (UPC), and he previously studied at the University of Crete. His research interests span hardware design and low-level software for real-time systems and embedded accelerators. He is the co-PI of the GPU4S (GPU for Space) ESA-funded project. Contact him at leonidas.kosmidis@bsc.es.

Hamid Tabani is a postdoctoral researcher at Barcelona Supercomputing Center (BSC). His research interests include computer architecture, autonomous driving, and machine learning. Tabani received a PhD in computer architecture from UPC. Contact him at hamid.tabani@bsc.es.

Carles Hernandez received the PhD in computer sciences from Universitat Politècnica de Valencia in 2012. He is currently senior PhD Researcher at BSC. His area of expertise includes time-predicable and reliability-aware processor design. Contact him at carles.hernandez@bsc.es

Jaume Abella is a Senior PhD Researcher at BSC since 2009. He holds a PhD in Computer Science from UPC (2005) and worked at Intel Corporation until 2009. His research interests include timing and functional verification of critical real-time systems, and performance analysis. He is member of IEEE and HiPEAC. Contact him at jaume.abella@bsc.es.

Francisco J. Cazorla is the director of the CAOS research group at BSC and a researcher at IIIA-CSIC. He has a PhD in computer science from UPC. His areas of interest cover hardware design and performance analysis of embedded real-time and high-performance systems. Contact him at francisco.cazorla@bsc.es.