BRASIL: A High-Integrity GPGPU Toolchain for Automotive Systems

Matina Maria Trompouki*, Leonidas Kosmidis†,*
*Universitat Politècnica de Catalunya †Barcelona Supercomputing Center (BSC)

***Abstract*—Embedded General Purpose Graphics Processing Units (GPGPUs) are increasingly used in automotive to enable Advanced Driving Assistance (ADAS) and Autonomous driving. However, their functional safety certification has been addressed so far only at the application or hardware level. In this work, we demonstrate how the entire GPGPU software stack can be developed and certified up to ASIL-D, including its toolchain. We introduce BRASIL, a GPGPU toolchain targeting high-integrity GPGPU computing and we explain our methodology to achieve ISO 26262 compliance, as well as our tool qualification strategy.**

I. Introduction

Embedded GPGPUs are increasingly used in existing and upcoming vehicles to satisfy the high computational need of Advanced Driver Assistance Systems (ADAS) and Autonomous driving in the automotive sector, which is regulated by the existing automotive standard, ISO 26262 [1]. However, it is widely acknowledged that they have brought a significant disruption in the their certification process, creating several challenges and opportunities in the automotive domain [2][3].

A high-integrity safety-certified electrical or electronic system in the automotive domain needs to comply with ISO 26262 at all its constituent levels, from hardware up to the application software. At the early phases of the product's safety life cycle, hazard analysis is performed to identify potential hazards in the operation of the system, as well as an assessment of their corresponding risk. Based on this process, a corresponding Automotive Safety Integrity Level (ASIL) is assigned, ranging from the low ASIL-A to the highest ASIL-D.

At hardware level, there are only a few GPUs developed according to ISO 26262 like the ASIL-B certified Imagination Technologies' PowerVR GX6650 GPU found in Renesas' R-Car H3 SoC [4], future PowerVR GPUs which are designed to reach higher ASIL levels [5] and NVIDIA's Xavier SoC which is used in this work and is designed to reach ASIL-D.

On the other hand, qualified toolchains for GPGPUs **do not exist yet**. In fact, the programming models used for mainstream GPGPU programming rely on very low-level operations such as dynamic memory allocation and pointers, which complicate both the certification of the GPGPU programs, as well as the qualification of their compilers. A recent work, Brook Auto [6] simplified the certification of automotive GPGPU applications by removing these features from a CUDA-like language, but didn't cover the tool qualification issue.

In this paper, we demonstrate that the certification of the entire GPU stack for the automotive domain is possible. We make the following contributions: a) we argue that building on existing ASIL-D certified graphics drivers implementing safety-critical API standards is not only desirable, but it also provides the shortest path to GPGPU certification based on the reigning standard. b) we present BRASIL, a GPGPU toolchain based on Brook Auto, which can be qualified according to ISO 26262. c) for the first time in the literature, we show which modifications and evidence are required to increase the tool confidence of a GPGPU toolchain and we describe our tool validation plan. d) finally, we provide evidence for our high-confidence error detection capabilities and we experimentally show that the required changes do not harm performance, but instead lead to an improvement compared to Brook Auto.

II. ISO 26262 Tool Qualification

When a tool like a compiler is used in safety critical systems and its output affects the behaviour of the system needs to be *qualified*. In other words, enough evidence must be shown, that the use of the tool does not introduce errors which can create a risk. The amount of evidence depends on the Tool Confidence Level (TCL), which is assigned depending on the classes of Tool Impact (TI) and the Tool Error Detection Level (TD), using a process called *Software Tool Classification*.

The Tool Impact specifies whether an error in the toolchain results in functional safety violation (2) or not (1). On the other hand, the Tool Error Detection describes the probability (confidence) about whether such an error can be detected and can be high (1), medium (2) or low or unknown (3).

Depending on the TCL of the tool, which can be from 1 (lowest) to 3 (highest), and the ASIL of the target system, different ways to obtain evidence and qualify the tool are recommended. The selected methods for a particular tool are described in its Software Tool Qualification Plan. If the TI is 1, the TCL is 1 and therefore no additional evidence and tool qualification is required. However, when the tool impact is high, as the tool confidence is decreased, the required TCL is increased requiring increasingly more evidence. For a tool with TCL 3 in order to be used in a certified system up to ASIL-D, the processes for tool qualification are: a) confidence from prior use, b) evaluation of the tool's development process, c) validation of the tool and d) development of the tool according to safety standards. The first two are highly recommended by ISO 26262 for lower ASILs (A-B) and in a lesser extent for higher ASILs. For higher ASILs, the recommendations are inverted, highly recommending c) and d) and simply recommending a) and b).

The compilation of the Tool Documentation is another step in the qualification process, in which the functionalities of the tool are described, together with its intended use, the user manual and the installation guide. Moreover, it contains information about known issues and work arounds.

Finally, a qualified tool is accompanied by the Qualification Report, which is a set of tests, procedures and their outcome that provide the evidence required according to the Tool Qualification Plan and shows that the tool works as it is expected. The Tool Classification, Qualification Plan, Tool Documentation and the Qualification Report form the *Tool Qualification Kit* for a given tool.

III. State-of-the-art in GPU and certification

Next we review the state-of-the-art in safety-critical GPU software we use to build a fully certifiable GPGPU solution.

Brook Auto: Brook Auto [6] is both a language for GPGPU application development and a toolchain implementation. However, only the language part and the software written in it are amenable to certification. On the other hand, its toolchain is not qualified. Brook Auto is implemented as a source-to-source compiler, which transforms Brook Auto code into C++ for the CPU part of the application, and OpenGL ES 2 for the GPU part of the application. The compiler is written in C++ and the generated application code is linked to a runtime also written in C++, which interfaces with the OpenGL ES 2 API. The code relies on multiple class hierarchies with dynamic polymorphism in order to support several backends. However, the OpenGL ES 2 API and driver implementation on which Brook Auto relies is not safety certified. Moreover, the C++ implementation of the Brook Auto compiler and the runtime violate critical code guidelines like ISO 26262 and MISRA [7] due to the dynamic polymorphism. Finally, for the same reason the toolchain contains significant portions of code which are not used within the OpenGL ES 2 backend.

Safety Critical Graphics Standards: Despite the fact that general purpose embedded GPUs have been only recently employed in automotive, traditional embedded GPUs supporting only graphics operations have been used in the automotive and in other safety critical domains (e.g. avionics) for several years now, to support rick cockpit and dashboard applications.

In order to support the use of GPUs for graphics operations in certified systems, Safety Critical graphics standards have been developed [8] achieving the highest degrees of certification across all safety critical markets and their respective certification standards. In 2016 the latest version of the OpenGL SC 2.0 standard was introduced, supporting programmable GPUs. This standard was based on the predecessor OpenGL ES 2.0, which has been extensively used across the embedded GPUs, by defining a subset removing functionality in order to minimise the implementation cost and guarantee that the dynamic features that prevent certification are removed. As a result, OpenGL SC 2.0 has been approved by all the relevant regulatory bodies in critical systems, and as such is used in the stringiest certification levels, including DAL-A in avionics and ASIL-D in automotive.

IV. BRASIL

In this work we introduce BRASIL, a GPGPU toolchain that builds on existing technologies and can be qualified for use up to the highest ASIL. BRASIL is based on Brook Auto [6] and contains modifications in its implementation which allow its qualification. Our implementation is available at [9].

Next we first examine the modifications we performed and then we present our Tool Qualification Kit.

A. Qualification Oriented Modifications

ISO 26262 highly recommends the development of a qualified tool according to a safety standard. We performed several improvements in Brook Auto's implementation following recommendations of **3 functional safety standards**: ISO 26262 [1], MISRA C++ [7] and OpenGL SC 2.0 [8]. It is worth noting that according to ISO 26262 no functional safety standard can be fully applied in the development process of software tools, and it can only be applied in small toolchains, but not on large compilers based on gcc or llvm.

Brook Auto compiler follows an object-oriented design, with multiple back-ends: it supports code generation for single and multicore CPUs as well as many GPU graphics APIs. In order to reduce the qualification effort and cost, we only retained two backends: the single core CPU for redundancy-based software verification and the OpenGL ES 2 backend and their corresponding runtimes, as well as the minimal host code backend. We also identified and eliminated dead code to achieve full code coverage as suggested by ISO 26262 and added additional assertions to capture potential violations.

In order to achieve higher compliance with safety standards, we also addressed the issue of pointers, whose use is restricted by both ISO 26262 and MISRA. The initial version was heavily based on C-style, pointer-based string operations, which we replaced with safe C++ string object methods in BRASIL. Furthermore, employing the C++ pass by reference feature has helped to eliminate pointer usage in function calls.

Additionally, we have changed the original object-oriented design of the runtimes, which is based on runtime polymorphism. This feature violates several MISRA C++ rules and the rule that forbids the use of hidden control flow in ISO 26262, due to hidden pointer redirections through vtables in the generated code, which can result in a potential failure and affect the deterministic execution time behaviour of the software. We modified the code in order to ensure that it is decided statically, at compile time, which methods are called.

On the graphics API compatibility, in order to comply with the OpenGL SC 2 standard, we have modified the OpenGL ES 2 backend, so that it only uses the OpenGL SC 2 subset. The most significant change in that aspect was the offline compilation of the shaders. BRASIL implements a two step process: The BRASIL compiler generates both the source code of the kernel in the shader language which is used for verification, as well as its compiled version in GPU executable form, using the offline qualified compiler of the driver provider. Therefore, if any compilation errors are introduced by BRASIL, they are detected immediately.

B. BRASIL Tool Qualification

Software Tool Classification: For any compiler there is a possibility that it introduces errors that may violate functional safety. Thus BRASIL's Tool Impact is high (1) and therefore it requires qualification. For any compiler, it is also true that the probability to have high or medium confidence to detect an error introduced by the compiler is very low. Although we designed BRASIL so that its output can be validated by multiple means, we conservatively assume that this is also our case, so that the Tool Error Detection Level (TD) is 3.

Tool Qualification Plan: Since we target the highest automotive integrity levels, given that the Tool Classification assigned TI 1 and TD 3, for its qualification we have to rely less on confidence from prior use and the evaluation of the development process, and provide more evidence about its validation and its development according to safety standards.

Our modifications do not change the functionality of the tool compared to Brook and Brook Auto, but offer compliance with safety standards. These preexisting equivalent tools have been extensively used in the past by the scientific High Performance community as well as by the AMD's customers of Brook+. We have compiled a list of existing open source applications which used Brook and can be used for validation, since they also include equivalent CPU implementations. Therefore there is high confidence of the tool working correctly and a list of known problems and work arounds.

Regarding the evaluation of the development process, ISO 26262 is very vague in this aspect and acknowledges that it should not be applied in the full tool, but to an adequate and relevant subset of it. Our changes to a subset (2 backends) were designed and developed to fit the automotive V-model software development according to the Automotive SPICE [10]. In particular, our safety standards modifications changed the software architectural design, as well as the software detailed design and unit construction. Further to this, we enhanced the software unit verification with static kernel code checks and dynamic kernel execution redundancy, via comparison against the CPU version. Finally, we addressed software qualification, by providing all the required material and evidence.

Validation is one of the main pillars of BRASIL's qualification. Validation happens in many different levels, to increase confidence. First, it takes place at the compilation of the generated code, which is statically checked by glslangValidator[11] and compiled by qualified compilers for CPU and GPU. If BRASIL introduces any error, syntax or semantic, these tools will detect it. In this case, the user may refer to the documentation for the known issues and their solutions.

Checking for functional errors is achieved by using the extended regression suite inherited from its predecessors, which contains use cases with known program output, therefore their execution allows to compare the result. Moreover, the preserved single CPU backend allows each GPGPU kernel to be validated with both CPU and the predetermined output. The next validation level is applied with the AMD Brook+ SDK applications ported in [6], which provide an option for random input selection and a CPU validation feature, so that the BRASIL generated GPU code is double-checked against both the application's native CPU implementation and the generated kernel CPU implementation. Finally, the user can rely on this automatic CPU kernel generation to validate the GPU output of the automotive application either during the testing phase only, or even during the system deployment.

Therefore, it is clear that there is high confidence that any errors which may be produced by BRASIL can be detected.

Last but not least, the compliance of the tool according to safety standards is a strong feature of BRASIL, not typically found in other compilers due to their size.

Tool Documentation: Thanks to the extended prior use of the tool and given that we preserved its specification, we are able to reuse large amount of previous efforts of documenting the tool. This includes the official documentation of the Brook project at Stanford, then the large open source community at Source Forge [12], AMD's commercial effort with Brook+ [13], the latest additions from Brook Auto [6] and its extensions to support accelerator programming education on low-cost embedded systems such as the Raspberry Pi [14]. From this we remove the parts that are not applicable in BRASIL, mainly due to the elimination of backends.

The documentation contains information about the use of BRASIL, its installation guide, a user manual/tutorial, the example use cases and their explanation used also for its validation, the expected behavior under anomalous operating conditions and a list of the known issues of the compiler. Information about the error detection and validation provisions in the design process of the tool and their use is also provided.

Tool Qualification Report: The Qualification Report contains the results of the tool qualification. ISO 26262 is mainly a process-based standard and therefore its tool qualification cannot be quantified. Instead, results of qualitative assessments such as the prior use, the development process and compliance with standards are provided. For those cases, we report that our assessment has been deemed positive.

V. Evaluation

The experimental evaluation of BRASIL with the validation tests described in the Qualification Plan on two automotive platforms show an overall improvement in execution time thanks to our modifications. Although this is not very noticeable on NVIDIA Xavier due to its high-end CPUs, on the lower-end Imagination Technologies' GPU there was an improvement of 100m compared to the original Brook Auto implementation, using AMD's Brook+ SDK applications.

Moreover, the execution of all validation tests (regression tests, existing applications from prior use and Brook+ applications) are passed without any errors. For this process, the GPU execution output produced by the GPU BRASIL backend is compared against the CPU output of the CPU BRASIL backend, as well as the hand-written CPU reference implementation of the same algorithms in AMD's applications. Moreover, the generated code is passed by the glslangValidator and both the offline qualified compilers for CPU and GPU without any errors.

VI. Future Safety-Certified GPGPU Computing and Conclusion

Since currently only graphics standards have been certified for critical systems, they are the only option available today for GPGPU applications which need to be certifiable for the highest integrity level. However, performing GPGPU computing using graphics APIs in a hand written manner is a very complex and error prone task [6], which can have the opposite effect on the certification, because either the process cannot be proven to be correct or the certification cost is increased due to complexity. Consequently, a higher-level GPGPU toolchain has to be used, such as our BRASIL proposal, which requires to be qualified and we have shown how this can be achieved.

On the other hand, general purpose programming models like CUDA and OpenCL cannot be used in safety critical systems, since they violate fundamental features not permitted by the current automotive standards and development guidelines. In addition, their drivers require to support a large API, which is hard to certify even if their programming models were compatible with ISO 26262. Last but not least, their compilers are difficult to qualify, due to the generality and the multiple features supported by those languages.

Therefore, the possible paths for future adoption of CUDA or OpenCL in safety critical systems are:

a) the definition of safety critical subsets of these languages. This is the path taken by the Khronos Group, which is currently working on such APIs for OpenCL. However, the large number of industrial partners involved in such standardisation activities has shown that it takes significant time until consensus is reached. Moreover, even after the standard has been accepted, it takes significant time until implementations are ready and additional delays until their libraries get certified and their compilers qualified. Especially the issues identified by [6], indicate that if those features are preserved in the safety standard subset for compatibility with the high-performance versions, certification and qualification cost is going to be high.

b) the standards to be changed or relaxed in order to permit the certification-problematic features of these languages. [2] foresees such a relaxation for the equally challenging introduction of machine learning in automotive. However, even if such a relaxation takes place in automotive standards, it is unlikely that this will happen in fail-operational systems like avionics, which are much more conservative. In that case, the safety critical standards will be significantly diversified, as well as their GPGPU adopted solutions.

c) the adoption of safety critical Domain Specific Languages (DSLs) or model-based GPGPU tools, combined with certified driver implementations for CUDA and OpenCL, implementing only the features used by these high-level languages. Such an approach is taken by $HIPA^{cc}$ [15] which has been proposed for another safety-critical domain (medical) [16] although it is not qualified, and its qualification cost will be higher as it is based on LLVM. Also the latest version of Matlab can generate CUDA code for GPGPUs from Simulink models. However, such solutions cannot be considered as general purpose, since they can only be used for certain applications (e.g. imaging in the case of $HIPA^{cc}$ [15]) and they currently suffer from the absence of certified GPU drivers and qualified tools for CUDA and OpenCL, despite they can offer application certifiability, just as the case of Brook Auto [6], which is general purpose.

To sum up, even if in the future one of these possible paths for adoption of CUDA or OpenCL languages in safety critical systems may become reality, BRASIL at this moment and in the near-future appears as the fastest way to achieve full compliance with the current version of ISO 26262, including at the toolchain level.

Acknowledgments

This work has been partially supported by the Spanish Ministry of Science and Innovation under grant TIN2015-65316-P and the HiPEAC Network of Excellence. Leonidas Kosmidis is also funded by the Spanish Ministry of Economy and Competitiveness (MINECO) under a Juan de la Cierva Formación postdoctoral fellowship (FJCI-2017-34095). The authors would like to thank Dr. Ian Broster from Rapita Systems and Sergio Carretero from Airbus Defence and Space, Getafe, Spain for their insightful discussions on tool qualification in critical domains during the early stages of this work.

References

[1] International Organization for Standardization, *ISO/DIS 26262. Road Vehicles – Functional Safety*, 2009.

[2] S. Saidi et al., "Future Automotive Systems Design: Research Challenges and Opportunities: Special Session," in *International Conference on Hardware/Software Codesign and System Synthesis (CODES)*, 2018.

[3] S. Alcaide et al., "Safety-Related Challenges and Opportunities for GPUs in the Automotive Domain," *IEEE Micro*, pp. 1–1, 2018.

[4] Renesas, "R-Car H3: Better Computing Capabilities and Compliance with Functionality Safety Standard," 2015, https://www.renesas.com/kr/en/solutions/automotive/soc/r-car-h3.html.

[5] A. Girdler, "PowerVR Automotive," in *AESIN ADAS and HAV Seminar*, 2018, http://2pe5rtjld2w41m0dy17n5an1-wpengine.netdna-ssl.com/wp-content/uploads/2018/12/Andrew-Girdler-Imagination-Technologies-1055.pdf.

[6] M. M. Trompouki and L. Kosmidis, "Brook Auto: High-level Certification-friendly Programming for GPU-powered Automotive Systems," in *Design Automation Conference*, ser. DAC '18, 2018.

[7] Motor Industry Software Reliability Association , *MISRA C++:2008 Guidelines for the use of the C++ language in critical systems*, 2008.

[8] Khronos Group, "OpenGL SC Overview," 2016, https://www.khronos.org/openglsc.

[9] L. Kosmidis and M. M. Trompouki, "BRASIL," 2019, http://github.com/lkosmid/brasil.

[10] DA QMC Working Group 13 / Automotive SIG (Automotive Special Interest Group and Quality Management Center in the German Association of Automotive Industry, "Automotive SPICE Process Assessment / Reference Model Version 3.1," November 2017, http://www.automotivespice.com/fileadmin/software-download/AutomotiveSPICE_PAM_31.pdf.

[11] Khronos Group, "glslang: An OpenGL and OpenGL ES shader front end and validator." 2017, https://github.com/KhronosGroup/glslang.

[12] I. Buck et al, "Brook Subversion Repository," 2007, https://sourceforge.net/projects/brook/.

[13] AMD, "AMD Brook+ Subversion Repository," 2009, https://sourceforge.net/projects/brookplus/.

[14] M. M. Trompouki and L. Kosmidis, "Brook GLES Pi: Democratising Accelerator Programming," in *Proceedings of the Conference on High-Performance Graphics*, ser. HPG '18, 2018.

[15] R. Membarth et al, "HIPAcc: A Domain-Specific Language and Compiler for Image Processing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 1, Jan 2016.

[16] R. Membarth et al., "Generating Device-specific GPU Code for Local Operators in Medical Imaging," in *IPDPS*, May 2012.