Comparison of GPU Computing Methodologies for Safety-Critical Systems: An Avionics Case Study

Marc Benito*† Matina Maria Trompouki* Leonidas Kosmidis†*
Juan David Garcia‡ Sergio Carretero‡ Ken Wenger§

*Universitat Politècnica de Catalunya (UPC) †Barcelona Supercomputing Center (BSC), Spain
‡Airbus Defence and Space, Getafe, Spain §CoreAVI, Waterloo, ON, Canada

***Abstract*—Introducing advanced functionalities in safety-critical systems requires using more powerful architectures such as GPUs. However software in safety-critical industries is subject to functional certification, which cannot be achieved using standard GPU programming languages such as CUDA and OpenCL. Fortunately, GPUs are already used in certified critical systems for display tasks, using safety-certified solutions such as OpenGL SC 2.0.**

In this paper, we compare two state-of-the art graphics-based methodologies, OpenGL SC 2.0 and Brook Auto/BRASIL for the implementation of a prototype avionics case study. We evaluate both methods on a realistic industrial setup, composed by an avionics-grade GPU and a safety-certified GPU driver in terms of development metrics and performance, showing their feasibility.

I. Introduction and State-of-the-Art

Due to their critical nature, safety-critical computing systems use legacy hardware, designed more than a decade ago. Inevitably, their performance capabilities are very limited and cannot keep up with the demand for advanced functionalities.

Recent safety-related innovations in avionics include computationally intensive tasks such as automatic take-off and flights [4] [2], which require high performance architectures such as GPUs. However, the use of a GPU in critical systems as a general-purpose computational element creates several safety-related certification issues, which the regulatory bodies and the avionics industries alike are not yet ready to address.

Although each safety-critical domain has different requirements, most safety critical systems share some common characteristics. These systems are programmed using safe subsets of the C language such as MISRA C [10]. One of the most important restrictions found in these systems is related to the use of low level memory manipulation, such as pointers and dynamic memory allocation. This is imposed by both language subsets and safety standards such as the DO-178 [11] used in avionics, especially in the highest criticality software level, DAL-A, in which the systems in control of the aircraft belong.

Analysis of GPU programming languages like OpenCL and CUDA, has revealed that they are incompatible with software certification in the automotive domain [8] – a less stringent domain than avionics – since their programming model relies on the use of explicit memory manipulation. Unless there is a change in the regulatory basis, which is a very slow process especially in avionics, this serious issue prevents the use of GPUs for computing in safety critical production systems.

On the other hand, safety critical systems already rely on GPUs for critical information visualization. Modern cars feature dashboard displays instead of traditional gauges for fuel, speed and engine information. In aviation, glass cockpits featuring LCD screens have replaced analog navigation instruments and LED indicators, while latest generation aircraft feature touch screens [1] [3] for both display and input. These displays are driven by avionics-grade GPUs, which comply with the hardware requirements imposed by the corresponding certification guidelines [5] for airborne electronic hardware.

On the software side, these GPUs use safety critical graphics APIs specifically designed to comply with the requirements of the highest criticality levels of these systems, as described in their certification standards, like the DO-178 in avionics and ISO 26262 in automotive. More concretely, Khronos' standards OpenGL SC 1.0 and 2.0 – where SC stands for "safety critical" – define a subset of the OpenGL ES 1.0 and 2.0 respectively.

The existence of a safety-certified programming method for GPUs, enables the use of graphics-based general-purpose computing techniques explored extensively by research works in the 2000s [7] before CUDA and OpenCL, adapted to use embedded APIs [12]. Alternatively, general-purpose GPU languages which automatically generate OpenGL ES 2.0/SC 2.0 code have been developed and shown as appropriate solutions for safety critical GPU programming certification for the automotive domain [8] [9]. These languages, Brook Auto and BRASIL, implement a subset of the Brook language [6], whose implementation address certification and tool qualification needs respectively, according to the ISO 26262 standard.

In this work we assess the feasibility of these two academic proposals in an industrial context. We implement a prototype avionics application in both programming methodologies and we evaluate their development effort and their performance.

II. Prototype Avionics Case Study

Our prototype avionics application models a safety critical scenario of vision-based navigation in which the aircraft to perform a rendez-vouz with another object in mid-air. For confidentiality reasons we can only provide a high-level description of the application. The application consists of two parts, one graphics-based (visualization) and one compute-based, in which general purpose computation is applied. The visualization part displays a 3D model of the aircraft for situational awareness. The general purpose computation part processes an 1920×1080 color image obtained from a camera and performs the necessary image analysis calculations for identifying the exact position of the rendez-vouz object. The result is also displayed on a screen with resolution 1280×720 and refresh rate 60Hz.

We implement the visual part in OpenGL SC 2.0 and the general purpose computation part in both OpenGL SC 2.0 following the method of [12] as well as in Brook Auto/BRASIL, using the open source implementation of [8] [9]. The applica-

 https://doi.org/10.23919/DATE51398.2021.9474060

TABLE I
DEVELOPMENT METRICS FOR THE TWO VERSIONS OF THE APPLICATION.

Programming language	OpenGL SC 2	OpenGL SC 2 (general-purpose compute)	Brook Auto/BRASIL (general-purpose compute)
Development time (days)	17	9	2.5
LOC (approx)	1400	1200	160

tion is executed on an avionics-grade GPU, AMD E8860 using a commercial safety-critical certified OpenGL SC 2.0 driver.

III. EVALUATION AND CONCLUSIONS

We compare the two GPU computing methodologies in terms of development metrics and performance. Both versions produce bit-identical results and achieve the required functionality of the application, both in terms of functionality and performance, showing the feasibility of the evaluated methodologies.

Development Metrics: Table I shows the comparison between the OpenGL SC 2.0 and Brook Auto/BRASIL versions in terms of development productivity. The development time of the full OpenGL SC 2 application version required 17 days and approximately 1400 lines of code (LOC). The compute part of the application took more than half of the development time of the OpenGL SC 2 version and accounted for 85% of its code. This increased effort is expected since performing computations in graphics APIs by using the methods of [12] is much more complex to develop and debug than performing graphics operations such as visualization tasks.

On the other hand, the Brook Auto/BRASIL version of the general purpose compute part, was performed in 3.5× less time than the corresponding OpenGL SC 2.0 implementation, without any prior knowledge of the language and the same functionality was achieved with 7.5× less amount of code.

It is important to note that the overall cost reduction expected in an industrial environment is much larger that the reduced development cost, especially in the safety-critical domain, in which verification and certification accounts for the largest portion of the production cost. In particular, less amount of written code is translated to fewer bugs, faster debugging and faster verification and certification, since all of them scale linearly with the code size and programming language complexity.

Performance: Both versions meet the refresh rate of an avionics display, which is 60 frames per second (FPS), showing that they are both appropriate for our application. However, in order to get insights for the different trade-offs and performance under extreme conditions for each version, we performed the same experiments also with vsync disabled, so that the performance of the GPU implementation is not limited to the screen refresh rate. The detailed results are shown in Figure 1.

First we compare the performance of the full application. The OpenGL SC 2.0 version achieves 2800 fps compared to the full application with Brook Auto/BRASIL implementation of the compute part which achieves 1500 fps, so both implementations exceed by far the display refresh rate. However, the later version requires an additional GPU operation to draw the result of the compute part on the screen. Adding this additional GPU operation in the full OpenGL SC 2.0 provides 1600 fps, which shows that the overhead of Brook Auto/BRASIL is quite small.

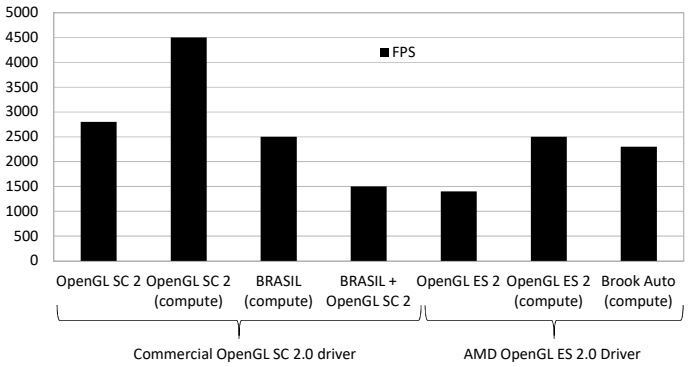

Fig. 1. Comparison between frames per second (FPS) performance of the different application versions without vsync on the AMD 8860 avionics GPU.

If we compare only the compute parts of the applications, the OpenGL SC 2.0 version achieves a very high performance of 4500 fps, while Brook Auto/BRASIL achieves 2500 fps. The reason is that the safety-critical driver we use is very latency optimised, pronouncing the overhead of BRASIL. We can see this if we execute the application on top of the stock Catalyst driver from AMD (last 3 bars of Figure 1): the difference of Brook Auto and the manual OpenGL ES 2 compute version is minimal. This is also evident from the fact that the full application in OpenGL ES 2 using the AMD driver has almost half of the performance of the full application in OpenGL SC 2 using the safety-critical driver, and slightly lower performance than the full application in BRASIL and OpenGL SC 2.0.

We conclude that both general-purpose computing methodologies can enable the certified use of GPUs in critical domains and can satisfy the performance requirements of advanced functionalities. Although manual implementation on OpenGL SC 2.0 can achieve higher performance, Brook Auto/BRASIL can provide a significant improvement in development effort.

ACKNOWLEDGMENTS

This work was funded by the Airbus TANIA-GPU Project ADS (E/200). It was also partially supported by the Spanish Ministry of Economy and Competitiveness under grants PID2019-107255GB and FJCI-2017-34095 and HiPEAC.

REFERENCES

[1] Airbus. Airbus begins deliveries of first A350 XWBs with touchscreen cockpit displays option to customers, 2019.
[2] Airbus. Demonstrates First Fully Automatic Vision-based Take-off, 2020.
[3] Boeing. Touchscreens come to 777x flight deck, 2016.
[4] Boeing. Autonomous Passenger Air Vehicle Completes First Flight, 2019.
[5] CAST-29. *Use of COTS Graphical Processors (CGP) in Airborne Display Systems*. Certification Authorities Software Team (CAST), 1997.
[6] I. Buck et al. Brook for GPUs: Stream Computing on Graphics Hardware. *ACM Trans. Graph.*, 23(3):777786, August 2004.
[7] J. D. Owens et al. A Survey of General-Purpose Computation on Graphics Hardware. *Computer Graphics Forum*, 26(1):80–113, 2007.
[8] M. M. Trompouki et al. Brook Auto: High-Level Certification-Friendly Programming for GPU-Powered Automotive Systems. In *DAC*, 2018.
[9] M. M. Trompouki et al. BRASIL: A High-Integrity GPGPU Toolchain for Automotive Systems. In *ICCD*, 2019.
[10] MISRA. *Guidelines for the Use of the C Language in Critical Systems*. Motor Industry Software Reliability Association, 2013.
[11] RTCA and EUROCAE. *DO-178B / ED-12B, Software Considerations in Airborne Systems and Equipment Certification*, 1992.
[12] M. M. Trompouki and L. Kosmidis. Towards General Purpose Computations on Low-end Mobile GPUs. In *DATE*, 2016.