Invited Paper: Assessing Unchecked Factors for Certification: An Experimental Approach for GPU Cache Parameters

Cédric Cazanove
ONERA, Toulouse, France

Benjamin Lesage
ONERA, Toulouse, France

Frédéric Boniol
ONERA, Toulouse, France

Jérôme Ermont
IRIT – INP – ENSEEIHT, Toulouse, France

Abstract

The certification objectives for airborne electronic hardware defined in AMC20-152A [9] and in AMC20-193 [8] capture some of the activities required for an applicant to embed a hardware platform in a safety-critical avionic system. For COTS (Commercially available Off-The-Shelf) platforms in particular, these objectives require applicants to identify functions, configuration settings, and resources present on the platform, and assess their use by the system. AMC20-152A however recognizes that documentation regarding the behavior of a COTS may be incomplete.

There is thus a strong push for applicants to the certification of a COTS to demonstrate their mastery of the platform, to highlight relevant *factors* (functions, settings, resources, etc.), and their use in their system. We outline in the following a standard approach to the exploration of unchecked factors of a platform, considering existing approaches in the literature, to build such a mastery. Our approach incrementally incorporates and validates knowledge of various factors by including them in micro-simulations compared to experimental ground truth.

2012 ACM Subject Classification Computer systems organization → Embedded hardware; Computer systems organization → System on a chip; Computer systems organization → Real-time system architecture; Computer systems organization → Multicore architectures

Keywords and phrases GPU, benchmarks, simulation, certification

Digital Object Identifier 10.4230/OASIcs.WCET.2024.3

Funding The work presented in this paper is part of the PHYLOG 2 project supported by the Directorate General of Civil Aviation (DGAC). It is funded by the French government through the France Relance program, based on the funding from the European Union through the NextGenerationEU program.

1 Introduction

Recent advances in machine learning allow the development of complex embedded functions which show a lot of potential for future algorithms in the avionic domain like ATTOL (Autonomous Taxi, Take-Off, and Landing). These algorithms have increasingly large computation requirements. This led to the emergence of a new generation of multi-core, hybrid architectures to offer greater computing power through increased compute parallelism. Nevertheless, hybrid platforms need to go through the same stringent certification process as other platforms before they are embedded in an avionic system. The European Union Aviation Safety Agency (EASA) defines Acceptable Means of Compliance (AMC) to guide the certification process. Each AMC defines a number of certification objectives the applicant must satisfy, but it does not prescribe actual tools and methods to that end. Architecture mastery, as an overarching certification requirement, needs to find an experimental method to assess and understand architectural factors.

Prior work has highlighted specific, sometimes previously unknown, factors on COTS platforms, and proposed methods to understand or capture their behaviour. Experimental results, and observations collected from the platform, often provide a ground truth or reference to assess factors. Mastering the complexity of COTS platforms may require dealing with multiple, complex interactions between these factors. As an example, the objective of computing the Worst-Case Execution Time (WCET) of a task on a COTS platform requires a mastery of all factors of said platform. We instead focus on understanding individual factors as a stepping stone towards more general models (WCET, simulator, etc.).

Contributions

This paper presents an experimental approach to reinforce knowledge and mastery of a COTS platform, more specifically to understand platform factors relevant to certification, i.e. used functions, configuration settings, shared resources, etc. We follow a widespread approach in practice: comparing expected behaviours (from the documentation or literature) to ones observed on the platform, for each factor or for increasingly large combinations thereof. The rationale is to provide supporting evidence for certification objectives related to these factors.

Where micro-benchmarks aim to support the characterisation of a platform by exercising specific factors, we propose the use of micro-simulators as reference models to compare against. The goal is to catalogue factors and related models, as micro-simulators and micro-benchmarks, to bootstrap the mastery of a new COTS platform for applicants. We focus the following on discovering the factors related to the cache hierarchy of an embedded GPU.

Organisation

The contribution is organised as follows. We first outline in Section 2 the certification objectives for COTS as per AMC20-152A and AMC20-193. Section 3 presents the overall approach to build a mastery of a complex COTS platform. The Jetson AGX Xavier is introduced in Section 4, as a supporting example of embedded COTS platform, alongside some of its factors up for validation as support for our evaluation (Section 6). The means of evidence for said factors are described in Section 5. We finally conclude by discussing related and future work (respectively in Sections 7 and 8).

2 Certification

As discussed in the introduction, certification is a key issue for safety-critical avionic systems. Processors involving several general-purpose cores and accelerators must undergo a stringent certification process before they are deployed. The European Union Aviation Safety Agency (EASA) and Federal Aviation Administration (FAA) respectively define Acceptable Means of Compliance (AMC) and Advisory Circulars (AC), setting down objectives that applicants to the certification process must satisfy. The joint A(M)C AMC20-193 and AMC20-152A in particular define objectives for the respective certification of hardware platforms and multi-core processors.

Overview of the AMC20-193. The AMC20-193 addresses the issue of multi-core platforms. It defines a Multi-Core Processor as a device with two or more activated processing cores, with a core being a device that executes software. It mainly focuses on both temporal and functional interference, that is, situations when two or more cores compete for shared resources. Interference may cause additional delays due to the arbitration of accesses to the resource or control flow variations due to external modifications of a shared variable. It may thus cause a loss of deterministic behaviour for the application. The impact of interference channels on applications in the system should be assessed. To address this issue, the AMC20-193 defines three main requirements: 1) the identification of the hardware and software resources of the processor, 2) the identification of their configuration settings (e.g., the L1 and L2 cache sizes, their replacement policy, if they are partitioned or not, etc.), and 3) the worse-case impact of interference in these resources.

Overview of the AMC20-152A. The AMC20-152A is complementary to the AMC20-193. It discusses the certification of complex COTS platforms. The clarifications proposed by AMC20-152A are important, as devices, especially COTS, become more complex and integrate in a single chip more functions and resources (such as GPUs) than older ones. The key objectives to certify COTS items according to AMC20-152A are 1) identifying the used function, 2) assessing the correct use of the COTS item, and 3) assessing the correct behaviour of the COTS item if used outside vendor specifications. AMC20-152A focuses on the risks inherent to the use of COTS, and that of incomplete or incorrect documentation. The issue when embedding COTS is the use of undefined or undocumented configurations, that may lead to unexpected behaviours.

From AMC20-193 and AMC20-152A we identified 4 activities for GPU platforms. **Activity 1:** It is necessary to master *complex* core architectures, that is, to identify all the configurations settings (e.g., resource capacities, arbitration policies, etc.) that are not clearly established and to assess them with certification evidence. To that end, we believe that stressing benchmarks would be needed in addition to documentation reviews. **Activity 2:** An assessment should be performed for each device of the platform. In particular, one should consider how the device is configured and accessed through hardware and software means, how it interacts with the rest of the system, and whether or not existing analysis techniques and tools apply. **Activity 3:** The utilisation of a COTS must be within the limit of the device manufacturer specification. This means that we need a specification of the COTS and its limits to check the compliance of usage. **Activity 4:** It is mandatory to qualify the COTS behaviour and all micro-code (e.g., the scheduling policies).

In this article, we focus on activity 1: architecture mastery for GPU platforms.

3 Mastering COTS platforms

Certification requirements, as discussed in Section 2, define a number of certification objectives for embedding a platform. Those revolve around the mastery of the platform, especially complex core architecture for COTS platforms. Industry practice and related work tend to rely on a similar approach, outlined in Figure 1, of iterating over identified factors to compare observed and expected behaviours. Discrepancies lead to refinements of the platform description and may guide further exploration. We consider the following activities:

- **Identify relevant platform factors** to capture functions, resources, and configuration settings available on the platform. The list may not be exhaustive and initially stems from vendor-provided documentation. Additional knowledge from the literature or the applicant's experience may also identify common factors to be assessed.
- **Identify unchecked factors** amongst the ones captured by the platform description. Each factor which has not been covered, or for which evidence is insufficient should be included. Even if documentation is available on the expected behaviour of a factor, it may be insufficient and benefit from further investigation.

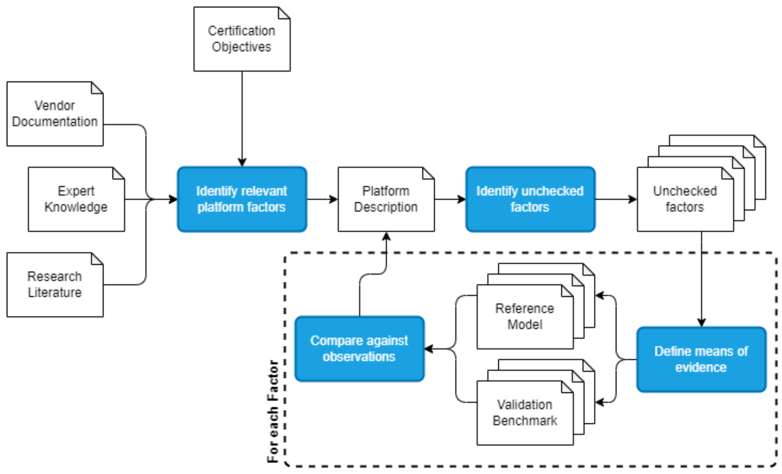

Figure 1 Overview of the process for mastering a complex COTS platform.

- **Define means of evidence** for each factor to provide evidence for certification, i.e. observe its behaviour, and capture expectations. This relies on the ability first to exercise the factor and capture relevant metrics, and second to model said factor under various configurations contending as capturing its behaviour. Models could take as an example the form of analytical models, simulations, etc.
- **Comparing against observations** should help discriminate the behaviour of the factor between possible candidates. It also provides evidence for certification that the factor is well understood. Discrepancies, however minor, may highlight unknown factors, or configurations which have yet to be described in the platform.

Our instantiation of the process relies in particular on the use of dedicated micro-simulators as a mean of collecting evidence of platform knowledge. Where possible, independent simulators and benchmarks should be defined to keep the complexity of the platform description manageable and to ease the argumentation for certification. The method will need a catalogue of benchmarks to validate and discover factors on a platform. Where dependencies exist, each simulator is built to include the impact of dependent factors as they are mastered by the applicant.

We illustrate the process in the remainder of the paper, focusing on the capturing parameters of the L1 cache of the Jetson AGX Xavier GPU, from platform description (Section 4), through evidence means (Section 5), to validation through comparison (Section 6).

4 Jetson AGX Xavier Platform

The Jetson AGX Xavier features the Xavier System-on-Chip (SoC). The Xavier SoC embeds a Volta GPU, an 8-core Carmel CPU complex, and dedicated accelerators for video and audio processing applications. Figure 2 presents a block diagram of the SoC as described in the platform Technical Reference Manual (TRM). This section presents the execution model of the Volta GPU to identify notable resources. The TRM provides little information on the Volta GPU architecture and related factors. We rely on supplemental vendor documentation and literature [10, 17, 3, 20, 16, 15, 7, 12], regarding the Volta architecture and its execution model in general. To highlight our approach we consider factors related to the GPU private L1 Data caches.

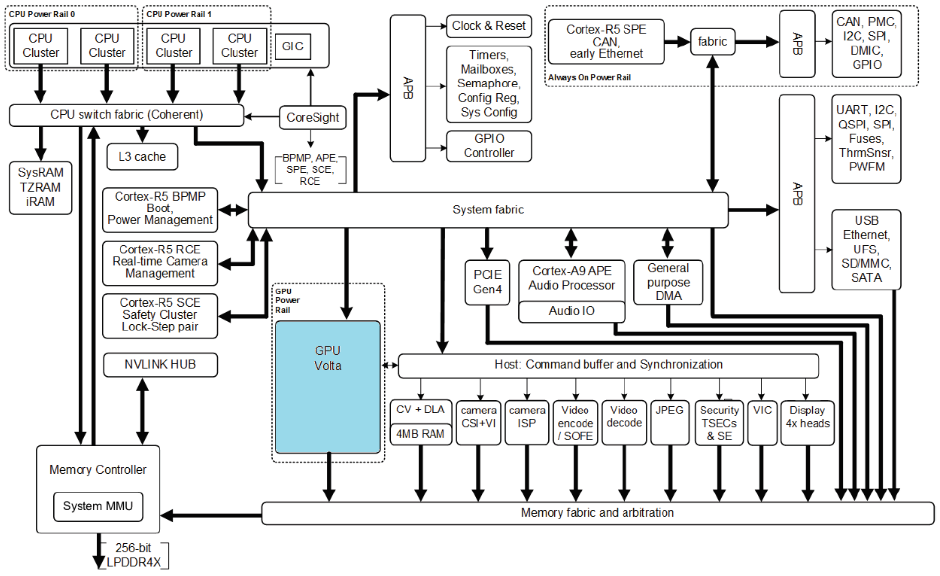
CPU Power Rail 0
CPU Power Rail 1
CPU Cluster
CPU Cluster
CPU Cluster
CPU Cluster
GIC
APB
Clock & Reset
Timers, Mailboxes, Semaphore, Config Reg, Sys Config
GPIO Controller
Cortex-R5 SPE CAN, early Ethernet
Always On Power Rail
fabric
APB
CAN, PMC, I2C, SPI, DMIC, GPIO
CPU switch fabric (Coherent)
CoreSight
BPMP, APE, SPE, SCE, RCE
L3 cache
SysRAM TZRAM iRAM
Cortex-R5 BPMP Boot, Power Management
System fabric
APB
UART, I2C, QSPI, SPI, Fuses, ThrmSnsr, PWFM
USB Ethernet, UFS, SD/MMC, SATA
Cortex-R5 RCE Real-time Camera Management
Cortex-R5 SCE Safety Cluster Lock-Step pair
NVLINK HUB
GPU Power Rail
GPU Volta
PCIE Gen4
Cortex-A9 APE Audio Processor
Audio IO
General purpose DMA
Host: Command buffer and Synchronization
CV + DLA
4MB RAM
camera CSI+VI
camera ISP
Video encode / SOFE
Video decode
JPEG
Security TSECs & SE
VIC
Display 4x heads
Memory Controller
System MMU
Memory fabric and arbitration
256-bit LPDDR4X

Figure 2 Overview of the Xavier SoC as described in vendor documentation [13].

4.1 Volta Execution Model

A user defines computation *kernels* for execution on the GPU, and other GPU operations through the CUDA Runtime or related high-level libraries, on top of the Runtime, such as cuDNN. We focus in the following on the definition and execution of computation kernels. A kernel call is shaped by the computation *grid* definition, i.e. the number of thread *blocks* and their shape. All thread blocks in a call are equally shaped [14]. The size of the kernel, the total number of threads invoked on a call, is thus the product of the number of threads per block and the number of blocks. Each thread is given a unique identifier to address different data segments. It is composed of its block index in the grid, and its thread index in the block. Both are accessible during kernel definition.

The block scheduler hierarchically dispatches each kernel block to a Graphics Processing Cluster (GPC), then picks a Thread Processing Cluster (TPC) on the GPC, and a Streaming Multiprocessor (SM) on the TPC. Dispatch depends on the block's resource requirements and the SM occupancy. The block will remain on its allocated SM until its threads are complete (unless a rescheduling occurs on preemption). The block's threads are further dispatched onto the SIMT Units (SU) which compose the SM. Each SU holds a register file and functional units to execute threads.

The Jetson AGX Xavier Volta GPU is composed of 8 Streaming Multiprocessors (SM), depicted in Figure 3, each with its own private L1 instruction and data caches (respectively ICache and DCache). SMs on the Jetson Volta GPU are partitioned into 4 TPC, each composed of 2 SMs. The GPU features a single GPC. All SMs share a unified L2 cache and 2 levels of TLB. Each SM is further divided into 4 SU. The register file on a SU, or L0 data cache, holds the context of multiple threads. Each SM is composed of 64 CUDA cores and 8 Tensor cores, split evenly across the SU. Tensor cores are a class of Deep Learning Accelerators supporting multiply-add operations on matrices, as instructed by the user (or a library). Restrictions on the number of registers in each SM imply that not all threads in a kernel call can execute concurrently. Blocks can hold a maximum of 1024 threads, at most 64 warps can reside at once on a single SM.

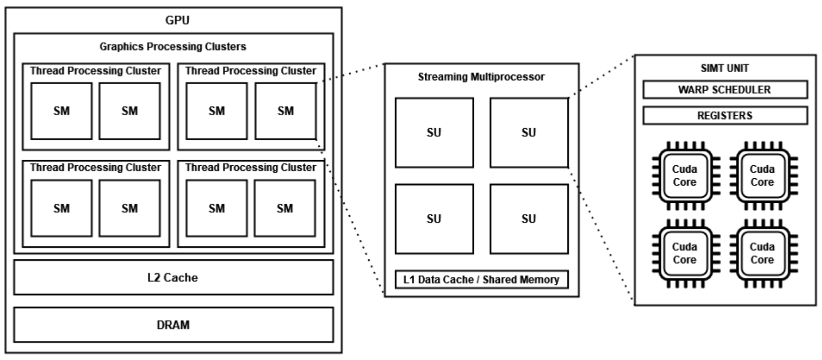

Figure 3 Overview of the Volta GPU architecture on the Jetson AGX Xavier.

SU follow the Single Instruction Multiple Thread (SIMT) execution model: the same instruction is executed across multiple threads at the same time, possibly addressing different data[1]. Threads are scheduled and executed in groups of 32, called a *warp*, such that each thread belongs to a single warp across its lifetime. Warp size and the allocation of threads in warps are not user-controlled parameters but platform-specifics. The warp scheduler on each SU can schedule up to 1 warp every cycle. Instructions from two distinct warps may coexist on the same core provided they rely on different functional units, e.g. a long-running load due to a cache miss and an integer addition. The combined register files on each SM can accommodate up to 64 warps to ensure the warp schedulers can maximise core occupancy.

4.2 DCache factors

The GPU embedded in the Jetson AGX Xavier relies on the Volta GPU architecture, and it is referenced as the GV10B model. While the GPU factors may be well documented at the architecture- or model-level, it is important to validate the documented factors as discussed in Section 2. There are furthermore factors in the GPU that are not documented and that an applicant needs to master to certify the platform, such as scheduling policies for the kernel components or the cache policy. We focus on the following factors related to the DCache private to each SM. Caches act as small buffers between a core and the comparatively slower SoC memory. They hold a copy of the most recently used data (and thus most likely to be reused) by a core. Table 1 presents a collection of cache-relevant factors, and their configuration as documented for the Volta GPU on the Jetson AGX Xavier. Bold factors are the ones considered in the subsequent evaluation.

5 Evidence means

To assess factors related to the DCache on the Volta GPU, we measure the behaviour of purpose-built characterisation benchmarks on the platform. We rely on a micro-simulator for the cache as our reference model. The micro-simulator allows us to explore different factors and their candidate configurations to find the best match for the observed behaviour.

[1] Note that the Volta GPU architecture introduces thread divergence, where threads may have different program counters. The scope and impact of thread divergence on the SIMT model are unclear.

Table 1 Selection of platform factors related to the private L1 DCache on the Volta GPU.

Factor	Description	Configuration
Line size	Number of (aligned) Bytes loaded upon a cache miss, i.e. when the accessed data is absent from the cache. Subsequent access to the same cache line may thus be served by the cache (hit).	32B [10]
Size	Number of cache lines held by the cache. Accesses to cache lines beyond the cache size will cause the evictions of older lines to make room for more recent ones.	128KB [11, 10]
Associativity	Number of candidates positions for a line in the cache. A cache line can only be stored in a limited number of places, based on its address. Simplifies the history required to maintain the ordering of cache lines.	1024 [10]
Replacement policy	Policy for selecting a position amongst the candidates based on the cache history upon inserting a line in the cache (miss). The policy also defines how the cache history is updated upon subsequent accesses (hit).	Not LRU [10]
Scratchpad allocation	Cache space allocated as a software-managed scratchpad for data shared between threads in a block. The accessible cache space is reduced as a result of scratchpad allocation.	8KB slices [12]
Prefetch policy	Policy allowing cache lines to be loaded in anticipation of future memory accesses. Which cache lines are prefetched depends on prior cache miss patterns.	N.A.
Coalescing policy	Policy for batching cache accesses caused by simultaneous thread execution into one or more cache accesses. Reduces the number of cache accesses required to serve a single warp.	N.A.
Write policies	Policies for dealing with data writes in the cache. Includes whether writes are blocking, cause insertion in the cache, and propagate all the way to the main memory.	N.A.
Inclusion policy	Captures how accesses, insertions, and evictions to (and from) the lower cache levels, e.g. the L2, affect the DCache.	N.A.
Coherency protocol	Policy capturing how concurrent accesses to the same data affect copies held in different private and shared, e.g. evictions, updates, etc.	N.A.

5.1 Experimental setup

We need to isolate the contribution of specific DCache factors on the GPU from other factors. To that end, we measure the L1 Cache Hit Rate during the execution of a single kernel running in isolation. The Cache Hit Rate captures the portion of accesses served by the cache over the total number of accesses performed by an application. It is thus mostly independent of the effects of the L2 factors. Focusing on a single-block, single-warp benchmark isolates observations from the highest level dispatch and scheduling policies (kernel, block and warp). It is still dependent on the scheduling of memory accesses by the SU, that is how threads within a warp are dispatched to the functional units.

On the Jetson AGX Xavier we use the Jetpack 5.10 to set a Linux environment of the platform. Concerning the Cuda version, we use Cuda 11.4 to develop our benchmark. Measurements are collected using Nvidia Nsight Compute (ncu).

5.2 Benchmark

It is necessary to exercise the DCache in a controlled fashion to understand its factors. CPU-based approaches [2] rely on configurable micro-benchmarks to generate a stream of memory access in which cache locality is known or behaviour is easy to predict. GPUs introduce additional instruction-level parallelism over CPU. We thus propose the following benchmark.

In a GPU several threads may execute concurrently and cause memory accesses. To discover the memory hierarchy of the GPU we introduce two notions: Step and Stride. Stride defines the distance in bytes between concurrent accesses of two consecutive threads; Step establishes the distance between two consecutive accesses from the same thread. All threads read from the same array. This allows us to define a simple read benchmark, per Algorithm 2, taking into account the instruction-level parallelism for GPU. By varying Step and Stride, we can control the way the threads of a kernel access the memory and the cache, and thus assess various parameters of the cache hierarchy. The implementation relies on an index-chasing pattern [2], with the pattern initialized as per Algorithm 1.

Algorithm 1 Array Initialization with Index Chasing.

```
1: Input: $n$                                   ▷ Size of the array
2: Input: $Step$                                ▷ Value of the Step
3: Output: $A$                                  ▷ Initialized array of size $n$
4: $A \leftarrow$ new array of size $n$
5: for $i \leftarrow 0$ to $n - 1$ do
6:     $A[i] \leftarrow (i + Step) \bmod n$
7: end for
```

Algorithm 2 Pseudo-Code of the Kernel of the read benchmark.

```
1: i ← threadIndex × STRIDES
2: for op ← 0 to NB_OP − 1 do
3:     i ← ptr[i]
4: end for
```

5.3 Simulation

As a reference model for the platform, we use a micro-simulator for the cache. The simulator focuses on implementing a single cache layer. It supports several policies and configuration points to explore possible factor configurations. We also build a surrogate model of our benchmark to mimic the sequence of memory accesses it generates on the platform under specific strides and step values. The generated sequence is fed into the simulator to estimate the L1 Cache Hit Rate.

This allows us to compare the output of the simulator and the benchmark. We compare our model of the platform under varied factor configurations with the measurements made on the real platform. The comparison allows to identify which configurations better represent the platform, and master undocumented factors as identified in Section 4. Where variations occur, we can refine the simulation configuration, or identify policies which better match the observed behaviours.

6 Evaluation

6.1 Assessing the DCache size

Configuration

We first aim to assess the DCache size. First, we need to isolate our experiment from the cache replacement policy effects. The Stride and Step parameters are set to prevent threads from reusing cache lines loaded by concurrent threads, within and between iterations. We thus set the Stride to the DCache Line Size (32B), with a 32B Line Size Step for each of the 32 threads in a warp, Step = 1024B. The accessed array is kept within the L2 cache size to limit the impact of inclusion and coherency policies. We use a read benchmark to remove the impact of write policies. As discussed in Section 5 focusing on a single-block single-warp kernel, we can mostly assume the memory accesses are issued in the order defined in the benchmark, from a single SU, so we will not have interference from an other SU. Moreover having a single-block single-warp kernel allows the focus on only one L1 Cache because the block will only run on one SM. We repeat the experiment with the kernel each time iterating over an increasingly bigger array. The DCache hit rate should suddenly drop whenever the array exceeds the cache capacity.

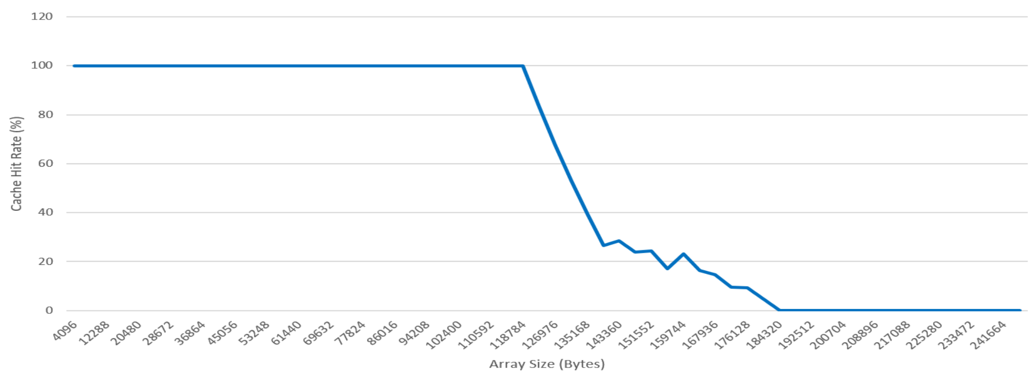

Figure 4 L1 Cache Hit Rate of a Read Benchmark with Chasing Index.

Observations

Figure 4 presents the observed DCache hit rates (y-axis) over an increasing array size (x-axis). We observe that the drop-off point does not correspond to the documented L1 size of 128KB, instead stalling when the array reaches a size of 116KB.

To validate our measurements, we considered a different collection tool (NVIDIA `nsys`). `nsys` does not support collecting Cache Hit Rates, but it did capture a drop-off point in execution time around 120KB. We think the NVIDIA instrumentation tools alocate space in DCache to collect measurements effectively reducing its capacity. However the 12 KB loss in cache size does not correspond to a valid scratchpad allocation, which operates in 8KB increment per the documentation.

6.2 Assessing of the DCache Associativity

Configuration

We repeat the same configuration as in the previous section. Beyond the drop-off point, the rate and shape of the hit rate decrease should provide information on the replacement policy and the associativity. To assess the DCache associativity, we consider different simulation configurations, namely varying the replacement policy and associativity to match configurations from the literature. Our simulator is configured using a 116KB DCache size as previously observed.

Observations

Figure 5 proposes a comparison of the observed hit rates (solid blue line) and simulated ones (dashed lines) while increasing the array size (x-axis). For simulations, we consider different configurations of associativity using the LRU or Random replacement policies. Under the current memory access pattern, the LRU, PLRU and FIFO replacement policies exhibit the same behaviour and we omit redundant results.

The random replacement policy results in an asymptotic behaviour with the hit rate gradually decreasing as the array size increases. This does not correspond to the relatively sharper observed drop. The 4-way LRU cache perfectly matches the observed drop initially, until the two diverge around 25% hit rate. The tail of the observed behaviour matches the slope of the 2-way LRU cache. This possibly hints at the use of adaptive replacement policies [18], a complex mechanism which may fit high-performance caches.

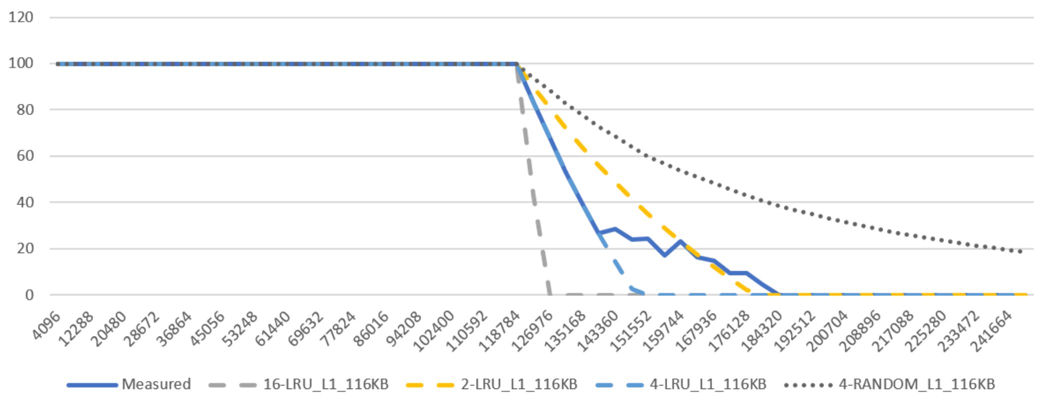

Figure 5 L1 Cache Hit Rate Simulation for several configurations.

7 Related work

With the increasing need and integration of GPUs in embedded systems, recent works have focused on understanding GPUs. There is a need to master said GPU to deploy Machine Learning in certified critical systems. NVIDIA represents today one of the leading GPU vendors in the market, including GPU for embedded systems. Hence there is consequent effort on mastering GPUs from NVIDIA such as the JETSON AGX XAVIER. However, one major problem persists: the GPU mainly works as a black box. Of course, some documented configurations are provided by the vendor but it is insufficient to fulfil certification objectives for critical systems. But thanks to prior works, configurations emerged from reverse-engineering or stressing benchmarks. Thanks to micro-benchmarking, [10, 6, 17, 3, 20, 16, 19, 4] discovered a number of factors of GPU platforms, especially regarding scheduling policies at different levels and related latencies. All of those works represent factors that applicants need to master for certification, as well as related means of evidence.

AMC20-193 recognises resource partitioning, as a suitable tool to alleviate specific objectives. In 2023, work by [6] pointed out the possibility of partitioning hardware resources of the most recent NVIDIA GPUs. The authors discovered the capability through public patent information as the feature is not documented by the vendor, its use poses additional challenges for certification with regards to AMC20-152A.

Other approaches aim to provide a model of the GPU as a whole, especially to predict its timing behaviour. This is crucial to WCET computation and related certification objectives. To the best of our knowledge, PasTiS [1] is one of the few efforts to build a GPU model suitable for WCET computation. While our approach does not aim to provide a model suitable for timing analysis, the platform description and knowledge derived from means of evidence do feed into the definition of such a model. Similarly, micro-simulators do not aim to replace full-fledged solutions such as GPGPUSim [5]. But they provide a quick reference.

8 Conclusion and future work

This paper presents an experimental approach to address the requirement for certification of the COTS platform according to AMC20-152A and AMC20-193. The approach, based on existing practices, aims to incrementally builds an accurate description of a platform. We suggest the use of micro-benchmarks to isolate each factor, and purpose-built micro-simulators to assess the behaviour of said factors. The approach thus benefits from existing work to capture factors to consider in a GPU, and the related means of generating evidence for certification. We illustrated the approach by considering the private cache level in an embedded COTS GPU. Our evaluation highlighted the applicability of existing work, for CPU-related factors, to the GPU characterisation.

As part of future work, we aim to catalogue factors identified in the literature, and reference the related micro-benchmarks and reference models, with the goal of easing the certification of existing or new COTS platforms. The work will include understanding how to collect observations from micro-benchmarks without impacting the platform behaviour or its configuration, or at the very least mastering the impact of the tools on the observations. We will continue the work on qualifying cache-related factors for embedded GPU, as well as factors related to instruction- and kernel-level parallelism.

References

1 Michaël Adalbert, Thomas Carle, and Christine Rochange. PasTiS: building an NVIDIA Pascal GPU simulator for embedded AI applications. In *11th European Congress on Embedded Real-Time Systems (ERTS 2022)*, 2022. URL: https://ut3-toulouseinp.hal.science/hal-03684680.

2 Alif Ahmed and Kevin Skadron. Hopscotch: a micro-benchmark suite for memory performance evaluation. In *Proceedings of the International Symposium on Memory Systems*, MEMSYS '19, pages 167–172, 2019. doi:10.1145/3357526.3357574.

3 Tanya Amert. *Enabling Real-Time Certification of Autonomous Driving Applications*. PhD thesis, The University of North Carolina at Chapel Hill, 2021. AAI28650154.

4 Tanya Amert, Zelin Tong, Sergey Voronov, Joshua Bakita, F. Donelson Smith, and James H. Anderson. TimeWall: Enabling Time Partitioning for Real-Time Multicore+Accelerator Platforms. In *2021 IEEE Real-Time Systems Symposium (RTSS)*, pages 455–468, 2021. doi:10.1109/RTSS52674.2021.00048.

5 Ali Bakhoda, George Yuan, Wilson Fung, Henry Wong, and Tor Aamodt. Analyzing CUDA workloads using a detailed GPU simulator. In *IEEE International Symposium on Performance Analysis of Systems and Software*, pages 163–174, 2009. doi:10.1109/ISPASS.2009.4919648.

6 Joshua Bakita and James H. Anderson. Hardware Compute Partitioning on NVIDIA GPUs. *2023 IEEE 29th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 54–66, 2023. URL: https://api.semanticscholar.org/CorpusID:259235797.

7 Alejandro J. Calderón, Leonidas Kosmidis, Carlos F. Nicolás, Francisco J. Cazorla, and Peio Onaindia. Gmai: Understanding and exploiting the internals of gpu resource allocation in critical systems. *ACM Trans. Embed. Comput. Syst.*, 19(5), September 2020. doi:10.1145/3391896.

8 EASA. AMC (Acceptable Means of Compliance) 20-193 on the use of multi-core processors (MCPs), 2020.

9 EASA. AMC (Acceptable Means of Compliance) 20-152A Development Assurance for Airborne Electronic Hardware (AEH), 2021.

10 Zhe Jia, Marco Maggioni, Benjamin Staiger, and Daniele Paolo Scarpazza. Dissecting the NVIDIA volta GPU architecture via microbenchmarking. *CoRR*, abs/1804.06826, 2018. arXiv:1804.06826.

11 NVIDIA. Jetson AGX xavier 32 GB specs. https://www.techpowerup.com/gpu-specs/jetson-agx-xavier-32-gb.c4088.

12 NVIDIA. Volta tuning guide. https://docs.nvidia.com/cuda/volta-tuning-guide/.

13 NVIDIA. *NVIDIA Xavier Series System-on-Chip: Technical Reference Manual.* NVIDIA Corporation, Santa Clara, California, April 2020.

14 NVIDIA. Nvidia heterogeneous computing on cuda platforms. https://docs.nvidia.com/cuda/cuda-c-best-practices-guide/index.html#heterogeneous-computing, 2022. Accessed: 2022-11.

15 Ignacio Sañudo Olmedo, Nicola Capodieci, Jorge Luis Martinez, Andrea Marongiu, and Marko Bertogna. Dissecting the CUDA scheduling hierarchy: a Performance and Predictability Perspective. In *2020 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 213–225, 2020. doi:10.1109/RTAS48715.2020.000-5.

16 Nathan Otterness, Ming Yang, Tanya Amert, James H. Anderson, and F. D. Smith. Inferring the scheduling policies of an embedded CUDA GPU. In *Workshop on Operating Systems Platforms for Embedded Real-Time Applications (OSPERT)*, 2017.

17 Nathan Michael Otterness. *Developing Real-Time GPU-Sharing Platforms for Artificial-Intelligence Applications.* PhD thesis, The University of North Carolina at Chapel Hill, 2022.

18 Moinuddin K. Qureshi, Aamer Jaleel, Yale N. Patt, Simon C. Steely, and Joel Emer. Adaptive insertion policies for high performance caching. In *Proceedings of the 34th Annual International Symposium on Computer Architecture*, ISCA '07, 2007. doi:10.1145/1250662.1250709.

19 Tyler Yandrofski, Jingyuan Chen, Nathan Otterness, James H. Anderson, and F. Donelson Smith. Making Powerful Enemies on NVIDIA GPUs. In *2022 IEEE Real-Time Systems Symposium (RTSS)*, pages 383–395, 2022. doi:10.1109/RTSS55097.2022.00040.

20 Ming Yang, Nathan Otterness, Tanya Amert, Joshua Bakita, James H. Anderson, and F. Donelson Smith. Avoiding Pitfalls when Using NVIDIA GPUs for Real-Time Tasks in Autonomous Systems. In *ECRTS*, 2018.