This is the title of the thesis

Daniel Leoncio Paredes Zevallos

A thesis presented for the degree of
Doctor of Philosophy

Supervised by:
Professor Louis Fage
Captain J. Y. Cousteau

University College London, UK
January 2015

I, AUTHORNAME confirm that the work presented in this thesis is my own. Where information has been derived from other sources, I confirm that this has been indicated in the thesis.

Abstract

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nam et turpis gravida, lacinia ante sit amet, sollicitudin erat. Aliquam efficitur vehicula leo sed condimentum. Phasellus lobortis eros vitae rutrum egestas. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Donec at urna imperdiet, vulputate orci eu, sollicitudin leo. Donec nec dui sagittis, malesuada erat eget, vulputate tellus. Nam ullamcorper efficitur iaculis. Mauris eu vehicula nibh. In lectus turpis, tempor at felis a, egestas fermentum massa.

Acknowledgements

Interdum et malesuada fames ac ante ipsum primis in faucibus. Aliquam congue fermentum ante, semper porta nisl consectetur ut. Duis ornare sit amet dui ac faucibus. Phasellus ullamcorper leo vitae arcu ultricies cursus. Duis tristique lacus eget metus bibendum, at dapibus ante malesuada. In dictum nulla nec porta varius. Fusce et elit eget sapien fringilla maximus in sit amet dui.

Mauris eget blandit nisi, faucibus imperdiet odio. Suspendisse blandit dolor sed tellus venenatis, venenatis fringilla turpis pretium. Donec pharetra arcu vitae euismod tincidunt. Morbi ut turpis volutpat, ultrices felis non, finibus justo. Proin convallis accumsan sem ac vulputate. Sed rhoncus ipsum eu urna placerat, sed rhoncus erat facilisis. Praesent vitae vestibulum dui. Proin interdum tellus ac velit varius, sed finibus turpis placerat.

Table of Contents

List of figures

List of tables

Abbreviations

API	**A**pplication **P**rogramming **I**nterface
JSON	**J**ava**S**cript **O**bject **N**otation

Chapter 1

Introduction

1.1 Motivation

Car manufactures want to reduce cost in terms of money and time required to develop, test and validate a new piece of software due to a change of supplier. For that reason, centralized end-to-end architectures are the solution they are aiming to, because for car companies such as BWW and Audi the car of future will be similar to a "data center on wheels" [1].

Centralized end-to-end architectures would be the first step stone toward to decoupling software and hardware [2]. This type of architectures not only will take advantage of internet connectivity, cloud computing and powerful heterogeneous processing units, but also will allow scalable, hierarchical and highly integrated system.

In other words, car manufactures prefer now a days low-latency, hierarchical and cost effectiveness of centralized end-to-end architectures, because today requirements of computational power, bandwidth, integration, safety and real-time [3].

However, car manufactures don't forget that at the end, in centralized end-to-end architectures, different types of software would run on top of an heterogeneous hardware supplied by companies such as NVIDIA, Mobileye or Qualcomm. Thus, it's important to analyze and understand how software

will behave under those conditions, in order to ensure a predictable and efficient system.

1.2 Industrial challenge WATERS 2019

Predictability is a key property for safety-critical and hard real-time systems [4]. Analyzing time related characteristics is an important step to design predictable embedded systems. However, in multi-core or heterogeneous systems based on centralized end-to-end architectures is harder to satisfy timing constrains due to scheduling, caches, pipelines, out-of-order executions, and different kinds of speculation [5]. Thus, development of timing-analysis methods for these types of architectures has become, nowadays, one of the main focus of research in both industry and academic environment.

Robert Bosch GmbH or Bosch, the German multinational engineering and electronics company, and one of the top leaders in development technology for the automotive industry announces every year *the WATERS Challenge*. The purpose of the WATERS industrial challenge is to share ideas, experiences and solutions to concrete timing verification problems issued from real industrial case studies [6].

This year, 2019, the challenge focuses on timing-analysis for heterogeneous software-hardware systems based on centralized end-to-end architectures. The platform chosen for this purpose is the NVIDIA® Jetson™ TX2 platform which has an heterogeneous architecture equipped with a Quad ARM A57 processor, a Dual Denver processor, 8GB of LPDDR4 memory and 256 CUDA cores of NVIDIA's Pascal Architecture. For the challenge it is available an Amalthea model for this platform to design a solution, and test it later on real hardware.

1.3 NVIDIA Jetson TX2: Architecture Overview

NVIDIA Jetson TX2 is an embedded system-on-module (SOM). It is ideal for deploying advanced AI to remote field locations with poor or expensive internet connectivity, Robotics, Gaming Devices, Virtual Reality (VR), Augmented Reality (AR) and Portable Medical Devices. In addition, it offers near-real-time responsiveness and minimal latency—key for intelligent machines that need mission-critical autonomy [7].

The main components of the Jetson TX2 are dual-core ARMv8 based NVIDIA Denver2, quad-core ARMv8 Cortex-A57, 8GB 128-bit LPDDR4 and integrated 256-core Pascal NVIDIA GPU. The quad-core Cortex-A57 and dual-core NVIDIA Denver2 can be seen as a cluster of heterogeneous multiprocessors (HMP) [8]. Both HMP and GPU shares a 8GB SRAM memory as shown in Figure 1.1. Hereafter, whenever we use the term **host**, we will refer to HMP, similarly we will use **device** to refer to GPU.

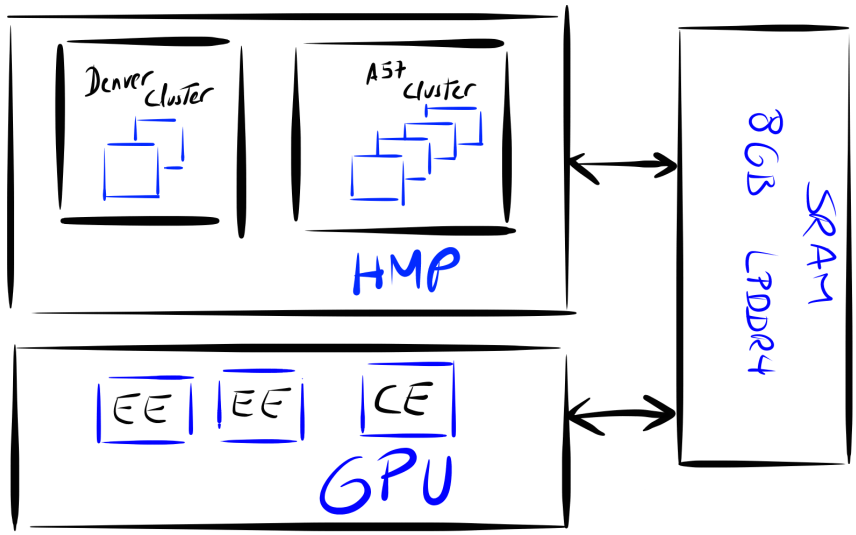

Figure 1.1: Jetson TX2 Architecture Overview

Any NVIDIA GPU has two types of engines, **Copy Engines** (CE) and **Execution Engines** (EE). The Jetson TX2 has only one CE and two EE also known as **Streaming multiprocessors**. CE is in charge of data transfers from host to device and viceversa. There is, moreover, the possibility that

EE and CE can run concurrently.

The GPU uses **streams** to run applications. The number of streams depends on the GPU resources. An application can run in one or multiple streams, the GPU scheduler, by default, manages how to application will be allocated on streams in order to maximize throughput. In Chapter 2, we will discuss in more detail how the TX2 GPU scheduler behave in case of multiple applications.

1.4 Jetson TX2 Amalthea Model

AMALTHEA is a platform for engineering multi- and many-core embedded systems. This platform enables the creation and management of complex tool chains including simulation and validation [9]. In the context of WATERS Challenge 2019, Bosch offers an AMALTHEA model of the Jetson TX2. In this model, a CPU runnable will read data from memory, execute some computation (Ticks) and write back data into memory as shown in Figure 1.2.

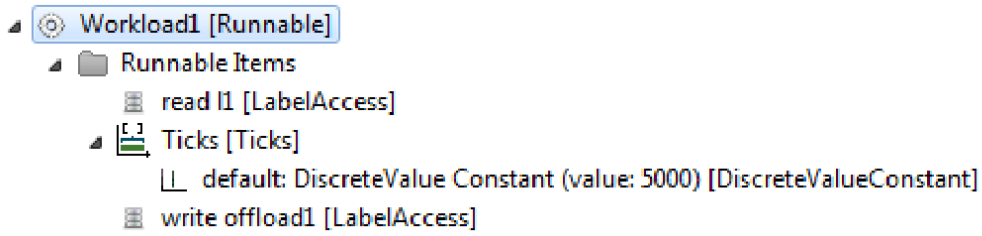

Figure 1.2: Runnable example for a CPU [6]

In the case of GPU modeling, the runnable will follow the same pattern as in the CPU case: read, execution, write back. However, the reading operation is actually to copy memory from host to device, thus it is modeled as *memory reading from host* and then as *memory writing to device*. On the other hand, the writing back operation requires to copy memory from device to host, therefore it is modeled as *memory reading from device* and then as *memory writing to host* as shown in Figure 1.3.

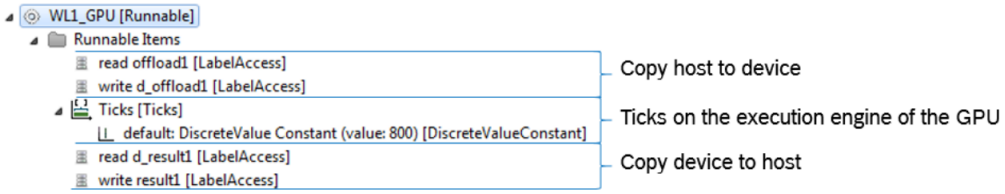

Figure 1.3: Runnable example for a GPU [6]

Chapter 2

CUDA and Jetson TX2

In this chapter...blablabla

2.1 NVIDIA GPU Software Model

Now a days computer applications run on heterogeneous hardware and GPUs are important in order to achieve high performance computing. Since 2006 running software on NVIDIA GPUs are known as a *CUDA application* [10]. A CUDA application will run concurrently multiple instances of special functions called **kernels**.Each instance runs on a **thread**. Moreover, these threads are arranged in **blocks**, and blocks compose **grids** as shown in Figure 2.1.

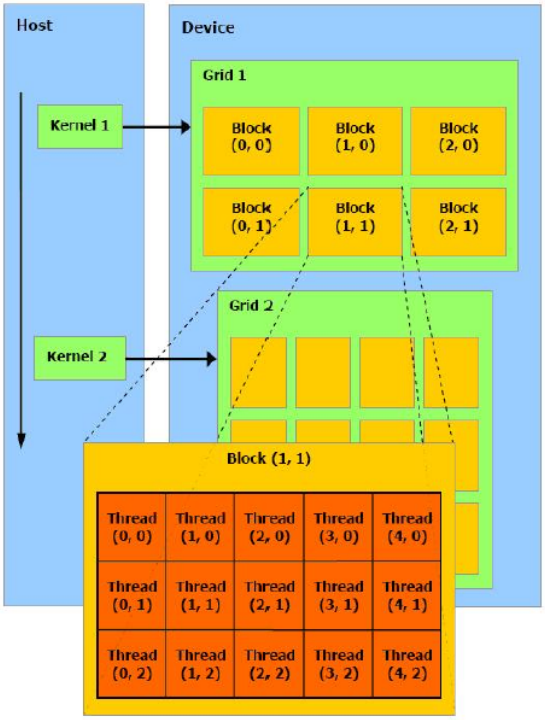

Figure 2.1: Organisation of grids, blocks, threads, and kernels [11].

It's logical to think that there is also a hierarchical memory structure. Threads, blocks and grids have access to different memory spaces as ilustrated in Figure 2.2. The types of memory are summarized in Table 2.1.

Table 2.1: Types of memories in a GPU

Memory	Main Characteristics	Scope	Lifetime
Global	R/W, Slow and big	Grid	Application
Texture	ROM, Fast, Optimized for 2D/3D access	Grid	Application
Constant	ROM, Fast, Constants and kernel parameters	Grid	Application
Shared	R/W, Fast, it's on-chip	Block	Block
Local	R/W, Slow as global, when registers are full	Thread	Thread
Registers	R/W, Fast	Thread	Thread

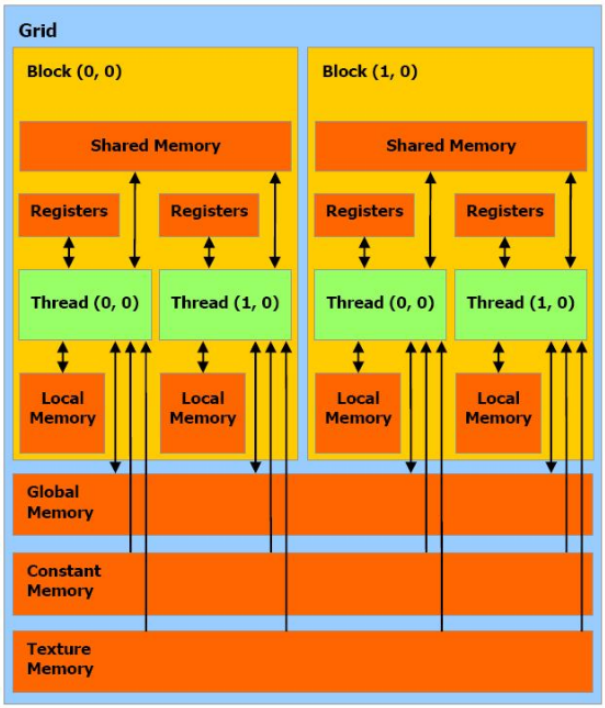

Figure 2.2: Memory hierarchy [11].

In summary, CUDA application solve problems that were modeled based on *divide and conquer* principle. Moreover, CUDA software model not only allow users to achieve high computational performance, but also CUDA application are highly scalable.

2.2 NVIDIA GPU Hardware Model

The CUDA architecture is based on **Streaming Multiprocessors** (SM) which perform the actual computation. Each SM has it own control units, registers, execution pipelines and local memories, but they also have access to global memory as ilustrated in Figure 2.3. A **stream** is a queue of CUDA operations, memory copy and kernel launch. We will talk more about streams in following sections.

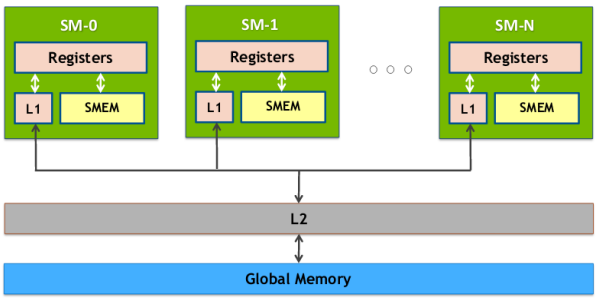

Figure 2.3: Memory hierarchy

When a kernel grid is launch blocks are enumerated and assigend to the SMs. Once the blocks are assigned, threads are managed in **wraps** by the **wrap scheduler**. A wraps is a group of 32 threads that run in parallel. Thus, it's highly recommendable to use block sizes of size $32N, N \in \mathbb{N}$, otherwise there would be "inactive" threads. A example is shown in Figure 2.4, where there is a block of 140 threads but since the wrap scheduler works with wraps, 20 threads are wasted and no other block can make use of them.

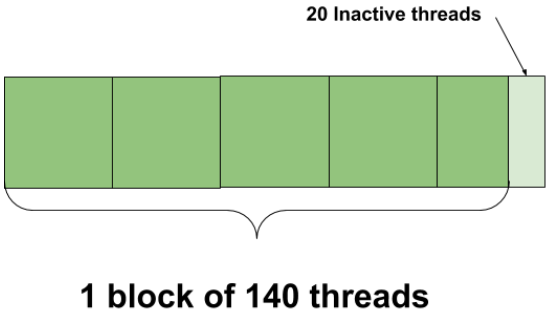

Figure 2.4: Inactive threads

The amount of threads and blocks that can run concurrently per SM depends on the number of 32-bit registers and shared memory within SM, as well as the CUDA computing capability of the GPU. Information related to maximum amount of blocks or threads, as well as the computing capability of the GPU can be display executing `deviceQuery` tool. Some information about Jetson TX2 is presented below:

```
CUDA Device Query (Runtime API) version (CUDART static linking)
Detected 1 CUDA Capable device(s)
Device 0: "NVIDIA Tegra X2"
  CUDA Driver Version / Runtime Version     9.0 / 9.0
  CUDA Capability:                          6.2
  Total amount of global memory:            7850 MBytes
  ( 2) SM, (128) CUDA Cores/SM:             256 CUDA Cores
  L2 Cache Size:                            524288 bytes
  Total amount of shared memory per block:  49152 bytes
  Total number of registers per block:      32768
  Warp size:                                32
  Max. number of threads per SM:            2048
  Max. number of threads per block:         1024
  Max dim. size of a thread block (x,y,z):  (1024, 1024, 64)
  Max dim. size of a grid size    (x,y,z):  (2^31-1, 65535, 65535)
```

2.3 NVIDIA Jetson TX2's GPU Scheduler

It's common to use several kernels in an application. In order to reduce computation time and maximaze GPU utilization, it's desire to run multiple kernels in parallel. CUDA uses streams to achieve this goal. As mentioned before, a stream is a queue of CUDA operations, memory copy and kernel launch. Thus, it is possible either to launch multiple kernels within one streams or multiple kernels on multiple streams. Operations within the same stream are managed in FIFO (First In First Out) fashion, thus, we will also use the term **stream queue** when we talk about FIFO queues within a stream. The Jeston TX2's GPU assigns resources to streams using its

internal scheduler.

Predictability is an important characteristic of safety-critical systems. It requires both functional and timing correctness. However, a detailed information about the Jetson TX2's GPU scheduler behaviour is not publicly available. Without such details, it is imposible to analyze timing constrains. Nevertheless, there are some efforts [12], [13] and [14] aimed at revealing these details through black-box experimentation.

NVIDIA GPU scheduling policies depend on whether the GPU workloads are launched by a CPU executing OS threads or OS processes. We will focus on the first case, because GPU computations launched by OS processes have more unpredictable behaviours, as stated in [12] and [13]. In this section, we will present GPU scheduling policies devired by [12] and use them in an example to clarify their use.

Let's start by defining some terms. When one block of a kernel has been scheduled for execution on a SM it's said that the block was **assigned**. Moreover, it's said a kernel was **dispatched** as soon as one of its blocks were assigned, and **fully dispatched** once all its blocks were assigned. The same applies to copy operations and CE.
There are, in addition, FIFO CE queues used to schedule copy operations, and FIFO EE queues used to schedule kernel launches. Stream queues feed CE and EE queues. Bellow we will present the rules that determine scheduler and queues behaviours.

- **General Scheduling Rules**:
  - **G1** A copy operation or kernel is enqueued on the stream queue for its stream when the associated CUDA API function (memory transfer or kernel launch) is invoked.
  - **G2** A kernel is enqueued on the EE queue when it reaches the head of its stream queue.
  - **G3** A kernel at the head of the EE queue is dequeued from that queue once it becomes fully dispatched.
  - **G4** A kernel is dequeued from its stream queue once all of its blocks complete execution.

- **Non-preemptive execution**:
  - **X1** Only blocks of the kernel at the head of the EE queue are eligible to be assigned.
- **Rules governing thread resources**:
  - **R1** A block of the kernel at the head of the EE queue is eligible to be assigned only if its resource constraints are met.
  - **R2** A block of the kernel at the head of the EE queue is eligible to be assigned only if there are sufficient thread resources available on some SM.
- **Rules governing shared-memory resources**:
  - **R3** A block of the kernel at the head of the EE queue is eligible to be assigned only if there are sufficient shared-memory resources available on some SM.
- **Copy operations**:
  - **C1** A copy operation is enqueued on the CE queue when it reaches the head of its stream queue.
  - **C2** A copy operation at the head of the CE queue is eligible to be assigned to the CE.
  - **C3** A copy operation at the head of the CE queue is dequeued from the CE queue once the copy is
  - **C4** A copy operation is dequeued from its stream queue once the CE has completed the copy.
- **Streams with priorities**:
  - **A1** A kernel can only be enqueued on the EE queue matching the priority of its stream.
  - **A2** A block of a kernel at the head of any EE queue is eligible to be assigned only if all higher-priority EE queues (priority-high over priority-low) are empty.

Authors in [12] mentioned that rules related to **registry resources** are expected to have exactly the same impact as threads and shared-memory rules.

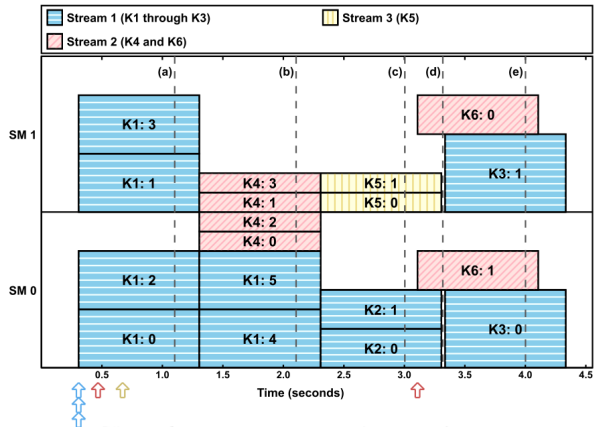

Figure 2.5: Basic GPU scheduling experiment [12]

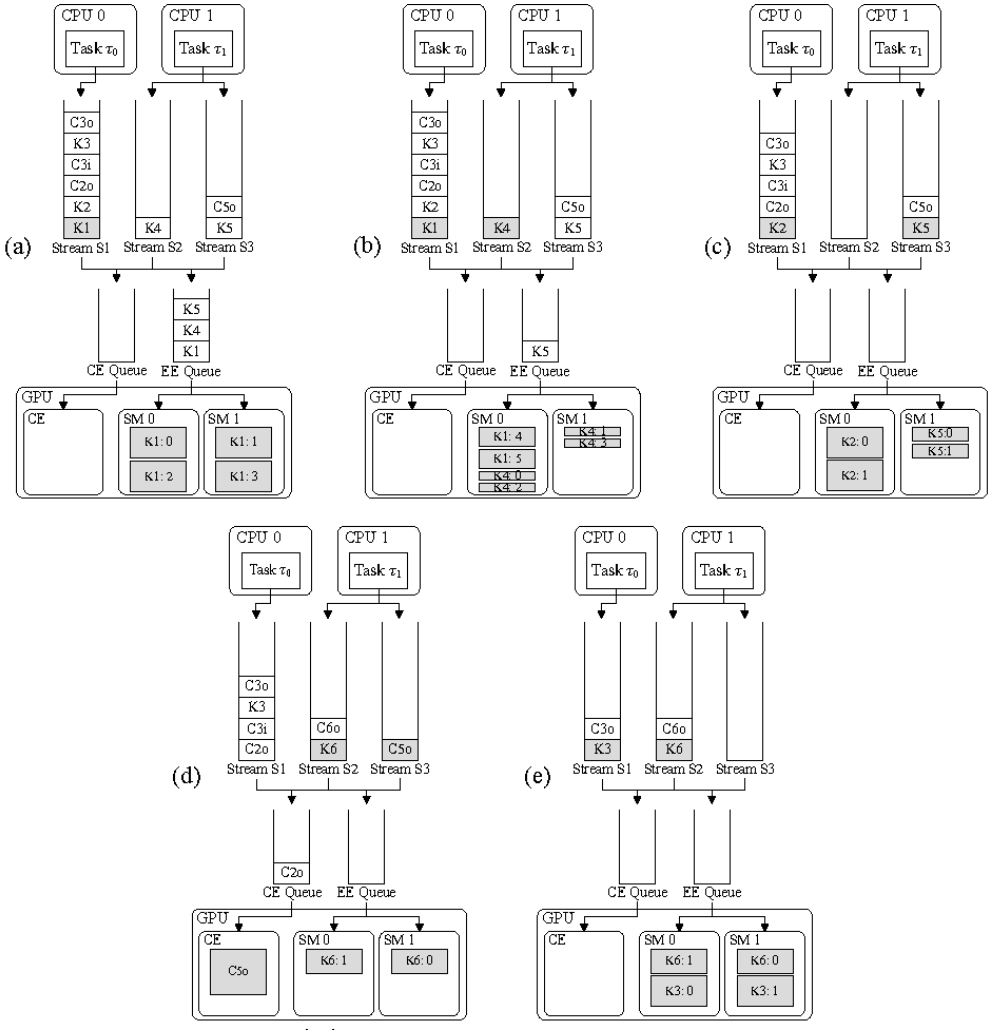

Figure 2.6: Detailed state information at various time points in Fig. 2.5 [12] .

Chapter 3

Jetson TX2's GPU scheduler response time analysis

3.1 Introduction

In this chapter, we present our approach to calculate the response time analysis for Jetson TX2's GPU scheduler based on the set of scheduling rules explained in the last chapter. **Need a better introduction**

3.2 Task model

There is a set of tasks or kernels τ of n independent kernels $\{\tau_1, \tau_2, \ldots, \tau_n\}$ on a single GPU. Each kernel has a period T_i defined as the separatin between two consecutivs releases of τ_i, thread execution time workload C_i and a grid of blocks g_i. Each block contains b_i threads.

$$\tau = \{\tau_i\}; \quad i \geq n \wedge n \in \mathbb{N} \tag{3.1}$$

$$\tau_i = \{T_i, C_i, g_i, b_i\} \tag{3.2}$$

Thus each kernel τ_i has a total of $g_i \cdot b_i$ threads, and the total execution time workload of τ_i is $C_i \cdot g_i \cdot b_i$. The utilization of each kernel is defined as the total execution time workload divided by the period, as stated in [15].

$$u_i = \frac{C_i g_i b_i}{T_i} \tag{3.3}$$

In addition, the total utilization of the set of tasks τ is defined as:

$$U_t = \sum_{\tau_i \in \tau} u_i \tag{3.4}$$

For a kernel τ_i we denote the release time as r_i, the completion time as f_i and response time as $R_i = f_i - r_i$ We assume that a kernel τ_i has a deadline equal to its period T_i.

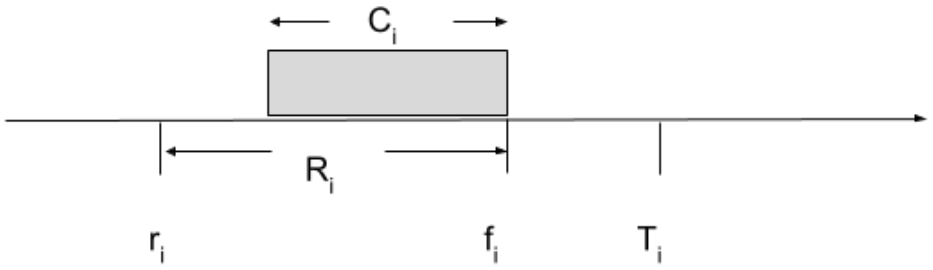

Figure 3.1: Time chart

3.3 Assumptions

For the calculation of the response time we have two assumption:

3.3.1 All blocks have the same amount of threads

The election of the optimum number of threads for a specific kernel is a hard task, for that reason there have been some efforts towards that direction [16], [17], [18], [19]. However, NVIDIA developers recommend, for practical

purposes, on their offical guides [20] and [21] to use block sizes equals to either 128, 256, 512 or 1024, because it has been documented that these values are more likely to take full advantage of the GPU resources. In our case we will assume that all the blocks, regarless the kernel, are the same size.

$$b_i = b, \quad \forall \tau_i \in \tau \tag{3.5}$$

3.3.2 One big streaming multiprocessor

This assumption is derived from the previous one. Each streaming multiprocessor in the Jetson TX2 has 2048 available threads and since b_i can be either 128, 256, 512 or 1024 ($2048/b_i = k, k \in \mathbb{N}$), we can think of the two streaming multiprocessors as a big one of 4096 threads.

It means that it could be allocated $2048/b_i$ blocks per SM or $4096/b_i$ blocks in the big SM. Hereafter we will refer the big SM as it were the only SM in the Jetson TX2's GPU. Thus, we defined g_{max} as the maximum number of blocks that can be allocated in the SM at some point in time.

$$g_{max} = \frac{b_{max}}{b}, \quad g_{max} \in \mathbb{N} \tag{3.6}$$

Where b_{max} is the maximum amount of threads in the GPU, in the case of Jetson TX2 is 4096.

3.4 Calculation of response time

In addition to the variables defined in our assumptions we define g_f as the number of blocks that are available at some point in time t, and t_a as the point in time in which a block $b_i \in g_i$ can be allocated.

For example in Figure 3.2a is shown that for a $t = t_1$ the amount of free blocks g_f is lower than g_{max} while in Figure 3.2b for a $t = t_2$, $g_f = g_{max}$.

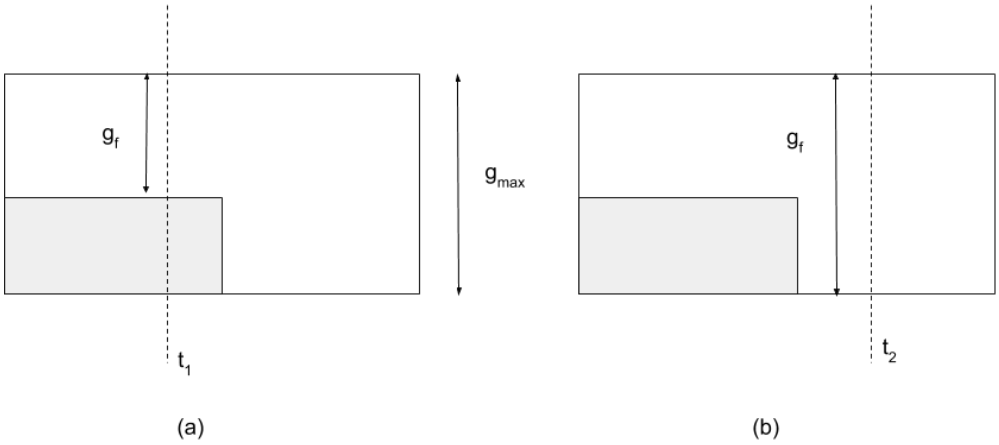

Figure 3.2: Free blocks (a) at $t = t_1$, $g_f < g_{max}$ (b) at $t = t_2$, $g_f = g_{max}$

In Figure 3.3 we present two cases. Let's assume there is a new kernel K4 which wants to allocate a block $b_i \in g_i$. In Figure 3.3a the release time r_4 of the kernel 4 is lower than t_1, which means that $t_a = t_1$ because $r_4 \leq t_1$ and kernel 3 (K3) was already dequeued. In Figure 3.3b r_4 lies between t_2 and t_3, in that case $t_a = r_4$, because all previous kernels were already dequeued and there are enough resources.

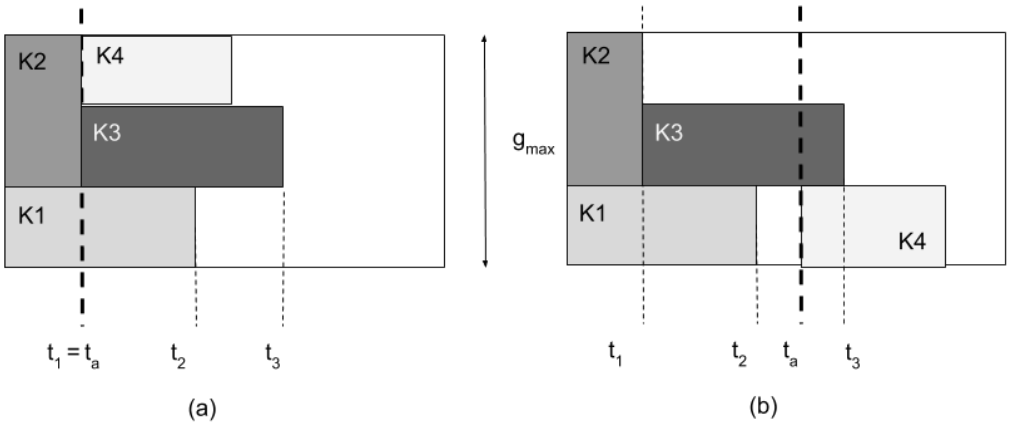

Figure 3.3: (a) $t_a = t_1 \quad \forall r_4$ s.t $r_4 \leq t_1$ (b) $t_a = r_4 \quad \forall r_4$ s.t $t_2 \leq r_4 \leq t_3$

Chapter 4

Research containing a figure

4.1 Introduction

This is the introduction. Sed vulputate tortor at nisl blandit interdum. Cras sagittis massa ex, quis eleifend purus condimentum congue. Maecenas tristique, justo vitae efficitur mollis, mi nulla varius elit, in consequat ligula nulla ut augue. Phasellus diam sapien, placerat sit amet tempor non, lobortis tempus ante.

4.2 Method

Donec imperdiet, lectus vestibulum sagittis tempus, turpis dolor euismod justo, vel tempus neque libero sit amet tortor. Nam cursus commodo tincidunt.

4.2.1 Subsection 1

This is the first part of the methodology. Duis tempor sapien sed tellus ultrices blandit. Sed porta mauris tortor, eu vulputate arcu dapibus ac. Curabitur sodales at felis efficitur sollicitudin. Quisque at neque sollicitudin, mollis arcu vitae, faucibus tellus.

4.2.2 Subsection 2

This is the second part of the methodology. Sed ut ipsum ultrices, interdum ipsum vel, lobortis diam. Curabitur sit amet massa quis tortor molestie dapibus a at libero. Mauris mollis magna quis ante vulputate consequat. Integer leo turpis, suscipit ac venenatis pellentesque, efficitur non sem. Pellentesque eget vulputate turpis. Etiam id nibh at elit fermentum interdum.

4.3 Results

These are the results. In vitae odio at libero elementum fermentum vel iaculis enim. Nullam finibus sapien in congue condimentum. Curabitur et ligula et ipsum mollis fringilla.

4.4 Discussion

Figure 4.1 shows how to add a figure. Donec ut lacinia nibh. Nam tincidunt augue et tristique cursus. Vestibulum sagittis odio nisl, a malesuada turpis blandit quis. Cras ultrices metus tempor laoreet sodales. Nam molestie ipsum ac imperdiet laoreet. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas.

Figure 4.1: RV Calypso is a former British Royal Navy minesweeper converted into a research vessel for the oceanographic researcher Jacques-Yves Cousteau. It was equipped with a mobile laboratory for underwater field research.

4.5 Conclusion

This is the conclusion to the chapter. Quisque nec purus a quam consectetur volutpat. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. In lorem justo, convallis quis lacinia eget, laoreet eu metus. Fusce blandit tellus tellus. Curabitur nec cursus odio. Quisque tristique eros nulla, vitae finibus lorem aliquam quis. Interdum et malesuada fames ac ante ipsum primis in faucibus.

Chapter 5

Research containing a table

5.1 Introduction

This is the introduction. Phasellus non purus id mauris aliquam rutrum vitae quis tellus. Maecenas rhoncus ligula nulla, fringilla placerat mi consectetur eu. Aenean nec metus ac est ornare posuere. Nunc ipsum lacus, gravida commodo turpis quis, rutrum eleifend erat. Pellentesque id lorem eget ante porta tincidunt nec nec tellus.

5.2 Method

Vivamus consectetur, velit in congue lobortis, massa massa lacinia urna, sollicitudin semper ipsum augue quis tortor. Donec quis nisl at arcu volutpat ultrices. Maecenas ex nibh, consequat ac blandit sit amet, molestie in odio. Morbi finibus libero et nisl dignissim, at ultricies ligula pulvinar.

5.2.1 Subsection 1

This is the first part of the methodology. Integer leo erat, commodo in lacus vel, egestas varius elit. Nulla eget magna quam. Nullam sollicitudin dolor ut ipsum varius tincidunt. Duis dignissim massa in ipsum accumsan imperdiet.

Maecenas suscipit sapien sed dui pharetra blandit. Morbi fermentum est vel quam pretium maximus.

5.2.2 Subsection 2

This is the second part of the methodology. Nullam accumsan condimentum eros eu volutpat. Maecenas quis ligula tempor, interdum ante sit amet, aliquet sem. Fusce tellus massa, blandit id tempus at, cursus in tortor. Nunc nec volutpat ante. Phasellus dignissim ut lectus quis porta. Lorem ipsum dolor sit amet, consectetur adipiscing elit.

5.3 Results

Table 5.1 shows us how to add a table. Integer tincidunt sed nisl eget pellentesque. Mauris eleifend, nisl non lobortis fringilla, sapien eros aliquet orci, vitae pretium massa neque eu turpis. Pellentesque tincidunt aliquet volutpat. Ut ornare dui id ex sodales laoreet.

Table 5.1: This is the table caption. Suspendisse blandit dolor sed tellus venenatis, venenatis fringilla turpis pretium.

Column 1	Column 2	Column 3
Row 1	0.1	0.2
Row 2	0.3	0.3
Row 3	0.4	0.4
Row 4	0.5	0.6

5.4 Discussion

This is the discussion. Etiam sit amet mi eros. Donec vel nisi sed purus gravida fermentum at quis odio. Vestibulum quis nisl sit amet justo maximus molestie. Maecenas vitae arcu erat. Nulla facilisi. Nam pretium mauris eu enim porttitor, a mattis velit dictum. Nulla sit amet ligula non mauris volutpat fermentum quis vitae sapien.

5.5 Conclusion

This is the conclusion to the chapter. Nullam porta tortor id vehicula interdum. Quisque pharetra, neque ut accumsan suscipit, orci orci commodo tortor, ac finibus est turpis eget justo. Cras sodales nibh nec mauris laoreet iaculis. Morbi volutpat orci felis, id condimentum nulla suscipit eu. Fusce in turpis quis ligula tempus scelerisque eget quis odio. Vestibulum et dolor id erat lobortis ullamcorper quis at sem.

Chapter 6

Final research study

6.1 Introduction

This is the introduction. Nunc lorem odio, laoreet eu turpis at, condimentum sagittis diam. Phasellus metus ligula, auctor ac nunc vel, molestie mattis libero. Praesent id posuere ex, vel efficitur nibh. Quisque vestibulum accumsan lacus vitae mattis.

6.2 Method

In tincidunt viverra dolor, ac pharetra tellus faucibus eget. Pellentesque tempor a enim nec venenatis. Morbi blandit magna imperdiet posuere auctor. Maecenas in maximus est.

6.2.1 Subsection 1

This is the first part of the methodology. Praesent mollis sem diam, sit amet tristique lacus vulputate quis. Vivamus rhoncus est rhoncus tellus lacinia, a interdum sem egestas. Curabitur quis urna vel quam blandit semper vitae a leo. Nam vel lectus lectus.

6.2.2 Subsection 2

This is the second part of the methodology. Aenean vel pretium tortor. Aliquam erat volutpat. Quisque quis lobortis mi. Nulla turpis leo, ultrices nec nulla non, ullamcorper laoreet risus.

6.3 Results

These are the results. Curabitur vulputate nisl non ante tincidunt tempor. Aenean porta nisi quam, sed ornare urna congue sed. Curabitur in sapien justo. Quisque pulvinar ullamcorper metus, eu varius mauris pellentesque et. In hac habitasse platea dictumst. Pellentesque nec porttitor libero. Duis et magna a massa lacinia cursus.

6.4 Discussion

This is the discussion. Curabitur gravida nisl id gravida congue. Duis est nisi, sagittis eget accumsan ullamcorper, semper quis turpis. Mauris ultricies diam metus, sollicitudin ultricies turpis lobortis vitae. Ut egestas vehicula enim, porta molestie neque consectetur placerat. Integer iaculis sapien dolor, non porta nibh condimentum ut.

6.5 Conclusion

This is the conclusion to the chapter. Nulla sed condimentum lectus. Duis sed tempor erat, at cursus lacus. Nam vitae tempus arcu, id vestibulum sapien. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus.

Chapter 7

Conclusion

7.1 Thesis summary

In summary, pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Nunc eleifend, ex a luctus porttitor, felis ex suscipit tellus, ut sollicitudin sapien purus in libero. Nulla blandit eget urna vel tempus. Praesent fringilla dui sapien, sit amet egestas leo sollicitudin at.

7.2 Future work

There are several potential directions for extending this thesis. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aliquam gravida ipsum at tempor tincidunt. Aliquam ligula nisl, blandit et dui eu, eleifend tempus nibh. Nullam eleifend sapien eget ante hendrerit commodo. Pellentesque pharetra erat sit amet dapibus scelerisque.

Vestibulum suscipit tellus risus, faucibus vulputate orci lobortis eget. Nunc varius sem nisi. Nunc tempor magna sapien, euismod blandit elit pharetra sed. In dapibus magna convallis lectus sodales, a consequat sem euismod. Curabitur in interdum purus. Integer ultrices laoreet aliquet. Nulla vel dapibus urna. Nunc efficitur erat ac nisi auctor sodales.

Appendix 1: Some extra stuff

Add appendix 1 here. Vivamus hendrerit rhoncus interdum. Sed ullamcorper et augue at porta. Suspendisse facilisis imperdiet urna, eu pellentesque purus suscipit in. Integer dignissim mattis ex aliquam blandit. Curabitur lobortis quam varius turpis ultrices egestas.

Appendix 2: Some more extra stuff

Add appendix 2 here. Aliquam rhoncus mauris ac neque imperdiet, in mattis eros aliquam. Etiam sed massa et risus posuere rutrum vel et mauris. Integer id mauris sed arcu venenatis finibus. Etiam nec hendrerit purus, sed cursus nunc. Pellentesque ac luctus magna. Aenean non posuere enim, nec hendrerit lacus. Etiam lacinia facilisis tempor. Aenean dictum nunc id felis rhoncus aliquam.

References

[1] "BMW and audi want to separate vehicle hardware from software." https://www.electronicdesign.com/automotive/bmw-and-audi-want-separate-vehicle-hardware-software.

[2] "End to end architecture." https://www.future-mobility-tech.com/en/technology/end-to-end-architecture.

[3] S. Kanajan, H. Zeng, C. Pinello, and A. Sangiovanni-Vincentelli, "Exploring trade-off's between centralized versus decentralized automotive architectures using a virtual integration environment," in *Proceedings of the conference on design, automation and test in europe: Proceedings*, 2006, pp. 548–553.

[4] T. A Henzinger, "Two challenges in embedded systems design: Predictability and robustness," *Philosophical transactions. Series A, Mathematical, physical, and engineering sciences*, vol. 366, pp. 3727–36, Nov. 2008.

[5] C. Cullmann *et al.*, "Predictability considerations in the design of multi-core embedded systems," *Ingénieurs de l'Automobile*, vol. 807, pp. 36–42, Sep. 2010.

[6] "WATERS 2019 – industrial challenge." https://www.ecrts.org/waters/waters-industrial-challenge/.

[7] D. Franklin, "NVIDIA jetson tx2 delivers twice the intelligence to the edge." https://devblogs.nvidia.com/jetson-tx2-delivers-twice-intelligence-edge/.

[8] *NVIDIA jetson tx2 system-on-module*. NVIDIA Corporation, 2014.

[9] E. APP4MC, "Project profile: Eclipse app4mc." http://www.amalthea-project.org/.

[10] NVIDIA, "Homepage - cuda zone." https://developer.nvidia.com/cuda-zone.

[11] *NVIDIA cuda c: Programming guide*. NVIDIA Corporation, 2010.

[12] T. Amert, N. Otterness, M. Yang, J. H. Anderson, and F. D. Smith, "GPU scheduling on the nvidia tx2: Hidden details revealed," in *2017 ieee real-time systems symposium (rtss)*, 2017, pp. 104–115.

[13] M. Yang, N. Otterness, T. Amert, J. Bakita, J. H. Anderson, and F. D. Smith, "Avoiding Pitfalls when Using NVIDIA GPUs for Real-Time Tasks in Autonomous Systems," in *30th euromicro conference on real-time systems (ecrts 2018)*, 2018, vol. 106, pp. 20:1–20:21.

[14] J. Bakita, N. Otterness, J. H. Anderson, and F. D. Smith, "Scaling up: The validation of empirically derived scheduling rules on nvidia gpus," *OSPERT 2018*, p. 49, 2018.

[15] M. Yang and J. Anderson, "Response-time bounds for concurrent gpu," *Proceedings of 29th Euromicro Conference on Real-Time Systems Work in Progress Session*, pp. 13–15, 2019.

[16] D. Mukunoki, T. Imamura, and D. Takahashi, "Automatic thread-block size adjustment for memory-bound blas kernels on gpus," in *2016 ieee 10th international symposium on embedded multicore/many-core systems-on-chip (mcsoc)*, 2016, pp. 377–384.

[17] R. Lim, B. Norris, and A. Malony, "Autotuning gpu kernels via static and predictive analysis," in *2017 46th international conference on parallel processing (icpp)*, 2017, pp. 523–532.

[18] Y. Torres, A. Gonzalez-Escribano, and D. R. Llanos, "Understanding the impact of cuda tuning techniques for fermi," in *2011 international conference on high performance computing simulation*, 2011, pp. 631–639.

[19] J. Kurzak, S. Tomov, and J. Dongarra, "Autotuning gemm kernels for the fermi gpu," *IEEE Transactions on Parallel and Distributed Systems*, vol. 23, no. 11, pp. 2045–2057, Nov. 2012.

[20] *NVIDIA cuda c: Best practices*. NVIDIA Corporation, 2019.

[21] *NVIDIA cuda c: Programming guide*. NVIDIA Corporation, 2019.