Addressing Analysis and Partitioning Issues for the WATERS 2019 Challenge

Daniel Casini, Paolo Pazzaglia, Alessandro Biondi, Giorgio Buttazzo, and Marco Di Natale
Scuola Superiore Sant'Anna, Pisa, Italy
Email: name.surname@santannapisa.it

***Abstract*—The WATERS 2019 Challenge seeks to find suitable models and analysis techniques for providing safe bounds to the response times of next-generation applications for autonomous-driving systems. This paper highlights the challenges arising from the scheduling of such workloads onto heterogeneous platforms, proposing modeling solutions and time-aware partitioning techniques aimed at minimizing the end-to-end latencies of processing chains.**

I. Introduction

The WATERS 2019 Challenge aims at finding models and analysis techniques tailored to the needs of next-generation applications for autonomous systems. Such software components are becoming more and more complex, and require high computational power. This need can be addressed by running these workloads on top of powerful heterogeneous platforms with multiple cores, graphical processing units (GPUs), and reconfigurable FPGAs [1], where cores are often capable of running at different processing speeds. The WATERS 2019 Challenge provides the *NVIDIA Jetson TX-2* as a reference platform. The Jetson TX-2 is composed of six cores, organized in two processor islands. The first provides four ARMv8 A57 cores running at 1.9 GHz, while the latter contains two 2 Ghz ARMv8 Denver cores. Each core (of both types) disposes of a 32KB L1 data cache and a 48KB L1 instruction cache. Internally to each island, cores share a 2MB L2 data cache. The platform is also provided with an iGPU (integrated GPU), composed of 256 CUDA cores grouped in two streaming multiprocessors (SMs), where each SM disposes of a 64KB L1 cache and shares a 512KB L2 cache with the other SMs.

From the application side, the challenge describes the requirements of a prototype for an Advanced Driver-Assistance System (ADAS), composed of 9 tasks, which cover the operations incurred by the whole computing path from the sensors input to the steer command. Pursuing the goal of providing the shortest possible worst-case response, computational activities (i.e., tasks) need to be properly mapped to the processor islands. Furthermore, some of them are also provided with a CUDA implementation, allowing to offload their computation to the GPU. Therefore, the challenge model gives rise to a non-trivial partitioning problem.

Contributions. The contribution of this paper is threefold. First, a scheduling model encompassing the requirements of the WATERS 2019 Challenge is presented. The proposed model is able to describe tasks that can experience different execution times depending on the type of core on which they are mapped, and to offload part of the computation on the GPU. The scheduling model is *intentionally simple* to match the specifications provided by the WATERS 2019 challenge. Second, a method for analyzing such tasks is presented. The offloading to GPUs is analyzed by means of the self-suspending task theory. Finally, the analysis is linearized to be included in a Mixed-Integer Linear Programming (MILP) formulation of the optimization problem, aimed at minimizing the end-to-end latencies of the processing chains by selecting the priority assignment, the task-to-core placement, and determining which tasks to accelerate on the GPU. Optimization results are discussed by comparing different trade-offs.

II. System Model

The model of this paper is based on the WATERS 2019 Challenge [2]. Our proposal for a formal characterization of the challenge model follows.

Platform Model. The WATERS 2019 Challenge Platform Model is based on the NVIDIA Jetson TX2, shown in Figure 1. This heterogeneous platform is composed of a quad-core 1.9GHz ARMv8 A57, a dual-core 2GHz ARMv8 Denver, and an integrated GPU. Consequently, the proposed system model comprises 2 types of processor cores $\mathcal{X} = \{x_1, x_2\}$. Each type x_i is characterized by a number of cores m_i, with $m_1 = 4$ (*A57* cores) and $m_2 = 2$ (*Denver* cores). We enumerate the cores with an index $k \in \{0, \ldots, m_1-1, m_1, \ldots, m_1+m_2-1\}$. We will then refer to the core with index k using the notation p^k. The integrated GPU is referred to with the symbol $\mathcal{G}$.

CPU Task Model. Tasks are executed on cores according to *partitioned fixed-priority preemptive scheduling*[1]. A task τ_i on a CPU is composed of an interleaved sequence of execution regions and *acceleration* regions, i.e., $\tau_i = \{R_i^1, \ldots, R_i^w\}$. Within each acceleration region, the CPU-task is not executing on a core, but *waits* for the completion of a GPU task that corresponds to an operation offloaded to the accelerator. For each task τ_i, a flag $\overline{a}_i$ denotes whether there exists an implementation that can be accelerated on $\mathcal{G}$. For the purposes of our work, we will consider the offloading to be always *synchronous* (a different interpretation of this term is used in the challenge description — see Section VI). Under this scheme, when a task offloads the computation to the GPU, it

[1]It is possible to configure a Linux-based system (as the one described in the challenge) with partitioned fixed-priority by assigning tasks to the SCHED_FIFO scheduling class and specifying affinities with the `sched_setaffinity()` system call.

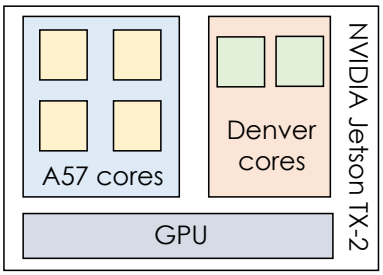

Figure 1. NVIDIA TX-2 platform abstraction

suspends its execution on the CPU, thus allowing the execution of other workload. This is in reality only one of the possible synchronization options offered by the CUDA API, but not necessarily the most efficient for solving the problem: we discuss about other possible synchronization options in Section VI. The offloading type $a_i \in \{\texttt{S}, \texttt{N}\}$ denotes if τ_i offloads its computations to the accelerator (`S` for synchronous) or not (denoted by `N`) – a set is used to easily extend the model to support other options for specifying the synchronization semantics in the future. If $a_i = \texttt{S}$, then τ_i is accelerated. Acceleration should refer to each region. However, since the challenge tasks only have one possible acceleration region, for simplicity, a_i refers to the entire task. When $a_i = \texttt{S}$, all accelerable operations of τ_i are offloaded to the GPU. If $a_i = \texttt{N}$, then task τ_i is entirely executed on a CPU. Clearly, if $\overline{a}_i = \texttt{FALSE} \Rightarrow a_i = \texttt{N}$.

Each j-th execution region is composed of an ordered sequence of n_i^j runnables $R_i^j = \{\rho_{i,1}^j, \ldots, \rho_{i,n_i^j}^j\}$, i.e., sequential portions of code. Each runnable $\rho_{i,s}^j$ is characterized by a worst-case execution time (WCET) $C_{i,s}^j(x_h^i)$, which depends on the type x_h^i of the core on which τ_i is allocated. Depending on the value of a_i, different operations can be needed, e.g., input data may need to be copied to/from the accelerator if $a_i \neq \texttt{N}$. Consequently, the list of runnables associated to a task τ_i may change depending on the value of a_i. The total duration of the j-th execution region of τ_i is denoted by $C_{i,j}(x_h, a_i)$ and is computed as the sum of the WCETs of all its runnables.

Conversely, the time that is required for processing the j-th acceleration region of τ_i, denoted by $A_{i,j}$, is not known a priori and depends on the response time of the computations executing on the GPU. An upper-bound to the duration of each acceleration region is provided in Section III. If $\overline{a}_i = \texttt{FALSE}$, the task τ_i is composed of a single execution region and no acceleration region exists. For convenience, the sum of all the lenghts of all the execution regions is defined as

$$C_i^k = \begin{cases} \sum_j C_{i,j}(x_h, a_i) & \text{if } \tau_i \text{ executes on } p^k, \\ 0 & \text{otherwise,} \end{cases}$$

where k is used to denote the core index. Conversely, the sum of all the time spent executing in the acceleration (modeled as suspension) regions is defined as $A_i = \sum_j A_{i,j}$.

Each task releases a potentially infinite sequence of job each separated by a minimum inter-arrival time T_i. Each job needs to complete within its relative deadline $D_i \leq T_i$, i.e., within D_i units of time from its release.

GPU Task Model. Due to lack of space, we omit a detailed description of the GPU scheduling mechanism: for a more comprehensive discussion, please refer to the WATERS 2019 Challenge description [2]. Under the challenge assumptions [2], each GPU task τ_i^G corresponds to a CPU task τ_j, i.e. it is possible to define a mapping $\mathcal{P}(\cdot)$ such that $\tau_j = \mathcal{P}(\tau_i^G)$. Each GPU-task consists of multiple segments, each one corresponding to the execution requirement needed by one of the acceleration regions of τ_j. The h-th segment of τ_i^G is characterized by a worst-case execution time $C_{i,h}^G$. For the weighted round–robin scheduling, each task τ_i^G is assigned to a time-quantum $Q_i \in [1, 500]\ ms$. Note that, according to our measurements, these assumptions provided with the challenge do not match the scheduling policy implemented by the Jetson TX2.

III. Response-time Analysis

This section presents a response-time analysis for tasks running on top of a heterogeneous platform, where each task starts and completes each instance on one of the CPU cores, but the computation can potentially be offloaded to the GPU multiples times during execution, i.e., in correspondence of acceleration regions. Consequently, there exists a one-to-one mapping between each acceleration region of a CPU-task and one of the segments of a GPU task. However, the duration of each acceleration region is unknown, and depends on the response time of the corresponding task executing on the GPU. To solve this issue, a response-time bound for GPU tasks is proposed next.

A. Response-Time Analysis for GPU Tasks

As discussed in Section II, GPU tasks are assigned to a time quantum Q_i and scheduled according to a weighted-round robin scheduling policy. Each tasks is allowed to execute up to Q_i consecutive time units, then it is preempted and another task τ_j can execute for at most Q_i units. GPU tasks are served in a cyclic manner. If a job needs less than Q_i time units to complete or there are no ready jobs of τ_i when its time-slice becomes available, the task is preempted or the execution slot is skipped. Due to the synchronous offloading mechanism adopted on the CPU side, each CPU task can have at most one pending acceleration at a time. Under the assumption of constrained deadlines, note that at most a single segment of a GPU task is executed in each time quantum. A *supply-bound function* $sbf_i(t)$ can be used to model the minimum amount of service provided by the GPU scheduling algorithm to each task τ_i in any interval of time of length t. Such a function is shown in Figure 2 and (for $k \geq 1$) can analytically be expressed as:

$$sbf_i(t) = \begin{cases} (k-1)Q_i + t - t^*(k) & \text{if } t^*(k) < t \leq t^*(k) + Q_i \\ kQ_i & \text{if } t^*(k) + Q_i < t \leq t^*(k+1), \end{cases} \quad (1)$$

where, $t^*(k) = k\Delta_i + (k-1)Q_i$, $\Delta_i = \sum_{\tau_j \in \mathcal{G}^S : \tau_i \neq \tau_j} Q_j$, and $k = \left\lceil \frac{t - \Delta i}{\Delta i} \right\rceil$.

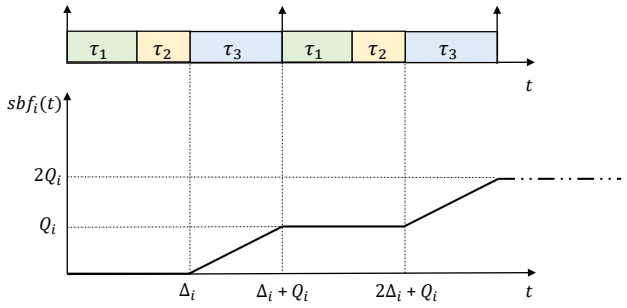

Figure 2. Supply-bound function for a task τ_i scheduled with the weighted-round robin policy.

Consequently, the response time R_i^G of a GPU task τ_i^G (if schedulable with a constrained deadline) is bounded by the least positive solution R_i^k of the following equation:

$$sbf_i(R_i^G) = C_{i,MAX}^G, \tag{2}$$

where $C_{i,MAX}^G = \max\{C_{i,h}^G \mid C_{i,h}^G \in \mathcal{C}(\tau_i^G)\}$.

If τ_i^G is the GPU task associated with a CPU task $\tau_j = \mathcal{P}(\tau_i^G)$, the length any of the acceleration regions of τ_j can be bounded by the worst-case response time of τ_i^G as computed with Equation (2). In the specific case of the task set provided with the challenge model, each CPU task has a single acceleration region. Once an upper-bound to the lengths of acceleration regions is available, the response-time of CPU tasks can be analyzed.

B. Response-Time Analysis for CPU Tasks

When the offloading is synchronous, CPU tasks can be analyzed as *self-suspending tasks* [3]. When a task offloads part of its computation to the GPU, it *suspends* on the CPU. Several analysis techniques are available in the literature (please refer to the work by Chen et al. [3] for a detailed review). Here, we briefly recall the approach in which the timing effects of suspensions of interfering tasks are accounted as *release jitter* [3], due to its fitness in being linearized and used in an optimization problem. In this model, acceleration regions correspond to suspension regions. Consequently, the response time of R_i^k of a CPU task τ_i running on core p^k is upper bounded by the least positive solution of the following recursive equation:

$$R_i^k = C_i^k + A_i + \sum_{\tau_s \in hp_k(\tau_i)} \left\lceil \frac{R_i^k + J_s^k}{T_s} \right\rceil C_s^k \tag{3}$$

where $hp_k(\tau_i)$ is the set of tasks with priority higher than τ_i allocated to p^k, $J_s^k = R_s^k - C_s^k$ if $a_s = S$ (i.e. if the task is performing offloading to the GPU), and $J_s^k = 0$ otherwise. A task set is deemed schedulabile if $\forall \tau_i, R_i^k \leq D_i$.

A more convenient formulation for the schedulability test is to check feasibility of a set of time intervals [4]:

$$\exists t \in \mathcal{S}_i : C_i^k + A_i + \sum_{\tau_s \in hp_k(\tau_i)} \left\lceil \frac{t + J_s^k}{T_s} \right\rceil C_s^k \leq t, \tag{4}$$

In [4] the set of check-points for the time interval is defined ad $\mathcal{S}_i = \{aT_s : \tau_s \in hp_s(\tau_i) \wedge a = 1, \ldots \lfloor T_i/T_s \rfloor\}$. However, Pazzaglia et al. [5] showed that a *sufficient-only* schedulability test with a very limited pessimism (less than to 2% of the considered cases) can be obtained by checking Equation (4) in a quadratic subset of the schedulability points in $\mathcal{S}_i$. The analysis can be extended to self-suspending tasks by using the subset of check-points for the case of scheduling with jitter in [5]:

$$\overline{\mathcal{S}_i} = \left\{aT_s - J_s^k : \tau_s \in hp_k(\tau_i) \wedge a = \left\lfloor \frac{D_i + J_s^k}{T_s} \right\rfloor \right\} \cup \{D_i\} \tag{5}$$

In this work, this simple and convenient approach has been exploited to encode an optimization problem aimed at finding solutions for the task-to-core and priority assignment that minimizes the end-to-end latencies of the processing chains contained in the challenge model.

IV. Optimization Problem

This section presents a formulation of the WATERS 2019 Challenge problem in the form of a MILP. The main objectives of the proposed formulation are the following:

- minimize end-to-end latencies of processing chains, according to the proposed objective function;
- select the most convenient task-to-core placement, accounting for both variable WCETs depending on the core type and the interference due to other tasks assigned to the same core;
- determine whether to accelerate tasks to find the most convenient trade-off between smaller WCETs occurring when a task is accelerated and longer acceleration regions when the GPU load is high (i.e., when multiple tasks are accelerated);
- optimize the priority assignment; and
- ensure schedulability, i.e., guarantee that each task completes within its deadline.

The linearization of this problem requires a solution to several non-trivial issues, which cannot entirely be discussed here due to lack of space. In this section, only the points explicitly related to the peculiarities of the problem are presented: the interested reader can refer to prior works (e.g., [6]–[8]) for hints about how some of the other constraints can be formulated. The proposed optimization problem is general enough for being applicable to a variety of task sets similar to the one included in the WATERS 2019 Challenge. For the sake of clarity, the following discussion directly refers to the challenge task set, as summarized in Table I (all time values in ms). The first part of the table reports the tasks parameters, by listing names, periods, and worst-case execution times, both when accelerated (field C_i^A) or not (field C_i^{NA}). The second part of the table reports the priorities, CPU mapping, and a_i values obtained by solving the proposed optimization problem. The following constraints make use of a numerical constant M to represent infinity, which is useful to code conditional constraints (a standard technique called *big-M*).

Table I
PARAMETERS AND SOLUTIONS OF THE OPTIMIZATION PROBLEM FOR THE TASK SET PROVIDED WITH THE CHALLENGE MODEL.

			A57		Denver		GPU	Solution		
ID	Name	T_i (ms)	C_i^{NA}	C_i^A	C_i^{NA}	C_i^A	C_i	PRIO	CPU	a_i
0	OS_Overhead	100	52,632	-	50	-	-	-	-	-
1	Lidar Grabber	33	14,379	-	10,868	-	-	5	0	N
2	DASM	5	1,958	-	1,3	-	-	7	2	N
3	CAN Polling	10	0,632	-	0,6	-	-	4	1	N
4	EKF	15	5,011	-	4,430	-	-	9	3	N
5	Planner	12	13,939	-	12,437	-	-	3	5	N
6	SFM	33	31,055	8,320	27,812	6,711	6,320	6	1	N
7	Localization	400	407,811	18,568	294,808	14,516	99,200	2	4	S
8	Lane Detection	66	53,732	8,667	42,238	7,626	21,867	8	4	N
9	Detection	200	-	4,958	-	4,086	92,800	1	1	S

GPU Offloading. A binary variable AC_i is introduced for each task τ_i to denote whether τ_i is offloaded to the GPU. The task set used in the challenge naturally imposes some constraints on this variable: **(i)** only the SFM, Localization, and Lane Detection tasks are provided with a both a CPU and GPU implementation, while **(ii)** the Detection task is *only* provided with a GPU implementation, i.e., $AC_i = 0$, for $i = 0, \ldots, 5$ and $AC_9 = 1$. For the other tasks, the MILP solver is free to select whether to accelerate the task (i.e., to select the value of AC_i, for $i = 6, 7, 8$).

Heterogeneous WCETs. The tasks provided with the challenge model are characterized by different execution times, depending on the type of core they are executed upon and whether they are accelerated. For each task, two variables ES_i and EN_i are used to model the execution time of task τ_i when $a_i = S$ (τ_i *is* accelerated) and $a_i = N$ (τ_i *is not* accelerated), respectively. The following constraints are enforced:

$$ES_i = \sum_{k=1}^{6} C_{i,k}^A \cdot P_{i,k}, \quad EN_i = \sum_{k=1}^{6} C_{i,k}^{NA} \cdot P_{i,k},$$

where $P_{i,k}$ is a binary variable set to 1 if task τ_i is allocated to core p^k, and $C_{i,k}^A$ and $C_{i,k}^{NA}$ are constants denoting the WCET of task τ_i when allocated to core p^k and is accelerated ($C_{i,k}^A$) or not accelerated ($C_{i,k}^{NA}$). Finally, a variable E_i is associated to each task τ_i to select the WCET value according to the value of AC_i, i.e., $E_i = EN_i$ if $AC_i = 0$, $E_i = ES_i$ otherwise. This condition is enforced by means of the following constraints:

$$E_i \geq EN_i - (1 - AC_i) \cdot M,$$
$$E_i \geq ES_i - AC_i \cdot M.$$

Length of acceleration regions. As discussed in Section III-A, the length of the acceleration regions of CPU tasks can be upper-bounded with the response time of the corresponding GPU tasks. Nevertheless, the analysis of Section III-A is based on the supply-bound function $sbf_i(t)$, which is highly non-linear and thus not suitable for being implemented in a MILP. Hence, we consider a simpler lower-bound for the supply-bound function, defined as [9]:

$$lsbf_i(t) = \max(0, \alpha_i(t - \Delta_i)) \leq sbf_i(t),$$

where Δ_i is defined as in Section III-A and $\alpha_i = \frac{Q_i}{Q_i + \Delta_i}$. This function is still not linear but can be more easily encoded in a MILP. By using $lsbf_i(t)$ and recalling that the response time of the longest segment of a GPU task τ_j^G upper bounds the length of any acceleration region A_i of the corresponding CPU task $\tau_i = \mathcal{P}(\tau_j^G)$, it is possible to rewrite Equation (2) as:

$$\alpha_i \cdot (A_i - \Delta_i) = C_{i,MAX}^G. \tag{6}$$

To implement Equation (6) as a MILP constraint we recall that the service delay Δ_i is defined as $\Delta_i = \sum_{\tau_j \in \mathcal{G}^S : \tau_i \neq \tau_j} Q_j$, but the set of tasks contained in $\mathcal{G}^S$ is a variable of the optimization problem. We introduce the auxiliary variable x and the constraint $x = \sum_{j=6}^{9} Q_j \cdot AC_j$, which accounts for the budgets of the tasks offloaded to the GPU (i.e., if $AC_j = 1$). With the definition of x and the corresponding constraint, it is possible to rewrite Equation (6) as:

$$A_i - (x - Q_i) \geq \frac{C_{i,MAX}^G}{Q_i} \cdot x - (1 - AC_i) \cdot M.$$

This constraint is linear only under the assumption that all budgets are constant and not optimized by the optimization problem.

OS Overhead. The challenge model provides an additional tenth task τ_0, not involved in any processing chain, to model the *overall* overhead due to the operating system. Unfortunately, no detailed information concerning the modeling of the OS overhead is provided with the challenge. For the sake of simplicity, we assume that an equal portion of τ_0 interferes on each core. Hence, a simple upper-bound to the interference generated by such a task on each core is given by $I_{OS}(t) = \left\lceil \frac{t}{T_0} \right\rceil C_{0,MAX}/m$, where $C_{0,MAX}$ is the maximum WCET of τ_0 (see Table I).

Objective function. We implemented our partitioning with an objective function that aims at *minimizing* the maximum end-to-end latency L_{MAX} of any processing chain. We recall

that the end-to-end latency of a set of tasks γ_x involved in a processing chain can be computed as [10]:

$$L_x = \sum_{\tau_i \in \gamma_x} (R_i + T_i) - T_{first}, \quad (7)$$

assuming that the external event triggering the chain arrives synchronously with the release of the first task $\tau_{first} \in \gamma_x$ of the chain.

V. Evaluation

The task set provided with the WATERS 2019 Challenge results to be unschedulable. Indeed, the utilization $U_5 = C_{5,}(x_h, a_5)/T_5$ of the Planner task (see Table I) is 1.036 and 1.16 if the task is scheduled on a Denver or A57 core, respectively. To make the problem feasible, WCETs have been scaled by a factor 0.8. The proposed MILP formulation has been solved with IBM CPLEX on a machine equipped with an Intel Core i7-6700K @ 4.00GHz. The solver is able to find the optimal solution within *one minute*. Table II summarizes the various task chains provided in the challenge model, which are graphically shown in Figure 3. The column 'Tasks' reports the IDs of the tasks included in each processing chain (see Table I for details). To mimic a pseudo-fluid (with respect to the bandwidth α_i) provisioning of service time, we set the value of the time quantum for each GPU task to 1 ms (i.e., to the minimum value allowed by the challenge specifications) for each accelerated task — as a first attempt, we were not interested in accounting for precise GPU scheduling issues due to the coarse-grained approximations mandated by the challenge model (see Section VI). As discussed in Section IV, the time quantum is a fixed parameter of the optimization problem. When considering the OS overhead as discussed in Section IV (task τ_0), the solver was not able to find a solution to the problem. This issue is due to the presence of some latency-sensitive tasks, i.e., tasks with very-low tolerance to additional delays. This is the case of the DASM task, which has a short deadline. To overcome this issue, we scaled the worst-case execution time of τ_0 by a factor $\gamma \in [0, 1]$. The minimum value $\gamma = 0.292$ leading to a feasible solution of the optimization problem has been found by binary-search: the corresponding solution is reported in Tables I, and II, in which all time values are expressed in ms. Lower values of the priority index correspond to higher priorities. CPU indexes in the range $[0; 3]$ belong to A57 cores, whereas the range $[4; 5]$ refers to Denver cores. To confirm our previous considerations, it is possible to observe from the solution reported in Table I that: **(i)** the latency-sensitive DASM task is allocated in isolation on core 2, and **(ii)** the Planner task, which has very-high utilization, is allocated in isolation on core 5. From Table I, we can observe that it is convenient to accelerate the Localization tasks (together with the Detection task): indeed, the gain in terms of WCET obtained by accelerating it is much higher than those obtained by accelerating the SFM and Lane Detection tasks. Table II reports the latencies (in ms) of the processing chains, computed as reported in Section IV. As expected, the longest chains are those involving the Localization task, which has a large period (see Equation (7)).

Table II
Latencies obtained for the various processing chains.

ID	Tasks	Latency (ms)
0	9 - 5 - 2	232,615
1	6 - 5 - 2	65,615
2	8 - 5 - 2	98,615
3	3 - 7 - 4 - 5 - 2	682,883
4	1 - 7 - 4 - 5 - 2	686,436
5	1 - 5 - 2	46,168
6	3 - 4 - 5 - 2	63,673
7	3 - 5 - 2	42,615

VI. Possible Extensions and Conclusions

This paper provides solutions for modeling, analyzing, and partitioning real-time applications running onto a GPU-enabled heterogeneous platform. The proposed approach represent a simple and flexible way for finding a suitable task-to-core mapping for real-time autonomous applications, and for deciding whether to accelerate tasks provided with both CPU-based and GPU-based implementations. Nevertheless, due to space issues and the problem complexity, we had to focus only on a subset of the problems arising from the WATERS 2019 challenge. Limitations and interesting directions for future research are discussed next.

Memory-aware analysis. Due to lack of space, the analysis proposed in this paper does not consider the effect of memory contention. To the best of our knowledge, the policy adopted by NVIDIA memory controllers to arbitrate the accesses to memory is currently undisclosed, although an empirical characterizations of the memory interference generated by concurrent accesses performed by CPU cores and the GPU is available in the literature [12]. A black-box experimentation aimed at determining how memory accesses are arbitrated (e.g., similar to the one presented by Amert et al. [13]) is a possible next research step towards memory-aware analysis techniques for heterogeneous platforms.

Due to the non-negligible memory space requirements of some components commonly found in autonomous applications (e.g., big deep neural networks) and the effects of memory placement on the timing behavior (for instance allocating them to the main memory or in the cache, e.g., by adopting locking techniques), an interesting future research line consists in analyzing the maximum memory space requirement [14] demanded by an autonomous real-time application executed upon an heterogeneous platform.

Analysis. Due to its flexibility in being coded as an optimization problem, this paper adopted a schedulability analysis for self-suspending task where suspensions are accounted as execution time for the task under analysis and as release jitter for interfering tasks. Future work should aim at improving the precision of the analysis implemented in our MILP-based

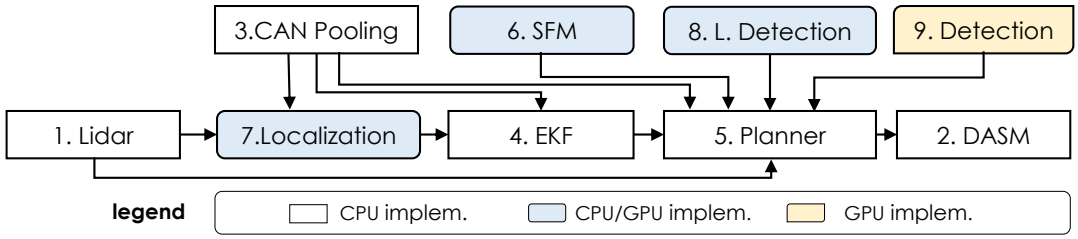

Figure 3. Processing chains of the WATERS 2019 Challenge. The number on the side of each task represent the task ID, imported from the challenge data.

partitioning algorithm, for instance integrating it with the analysis by Nelissen et al. [15].

Another interesting researching direction consists in analyzing the performance of the processing chains provided in the challenge model when implemented with a data-driven activation. With such a semantics, an instance of a task with data dependencies from other tasks is triggered only when dependencies are satisfied. New analysis techniques are needed to consider platform heterogeneity, e.g., extensions to the Compositional Performance Analysis [16, 17] (CPA) or to recent methods for analyzing parallel tasks under partitioned scheduling [18, 19] by means of self-suspending task theory.

GPU Modeling. According to our measures obtained by profiling the Jetson TX 2 platform, the round-robin scheduling performed by the Jetson TX2 does not seem to match the model proposed with the challenge. Indeed, scheduling does not seem to be performed in the time-domain but with the granularity of CUDA kernels. Furthermore, according to our knowledge of the CUDA API, synchronous offloading to the GPU still involves a suspension of the calling task (and not active waiting as discussed by the challenge). Richer task models, with respect to the self-suspending task model, are required to properly analyze asynchronous offloading.

Extension to ROS-based systems. The autonomous application presented in the WATERS 2019 Challenge assumes to be implemented as a standalone program, without using the facilities provided by *Robotic Operating System* (ROS). Nevertheless, due to the increasing use of ROS in the design, development and deployment of autonomous systems, the integration of the approach proposed in this paper with state-of-the-art techniques for analyzing ROS [20] systems represents an interesting direction for future research.

References

[1] A. Biondi, A. Balsini, M. Pagani, E. Rossi, M. Marinoni, and G. Buttazzo, "A framework for supporting real-time applications on dynamic reconfigurable fpgas," in *Proc. of the IEEE Real-Time Systems Symposium (RTSS 2016)*, December 2016, pp. 1–12.

[2] A. Hamann, D. Dasari, F. Wurst, I. Saudo, N. Capodieci, P. Burgio, and M. Bertogna. WATERS Industrial Challenge 2019.

[3] J.-J. Chen, G. Nelissen, W.-H. Huang, M. Yang, B. Brandenburg, K. Bletsas, C. Liu, P. Richard, F. Ridouard, N. Audsley, R. Rajkumar, D. de Niz, and G. von der Brüggen, "Many suspensions, many problems: a review of self-suspending tasks in real-time systems," *Real-Time Systems*, Sep 2018.

[4] J. Lehoczky, L. Sha, and Y. Ding, "The rate monotonic scheduling algorithm: exact characterization and average case behavior," in *[1989] Proceedings. Real-Time Systems Symposium*, Dec 1989.

[5] P. Pazzaglia, A. Biondi, and M. D. Natale, "Simple and general methods for fixed-priority schedulability in optimization problems," in *Proceedings of the International Conference on Design, Automation and Test in Europe (DATE 2019)*, March 2019.

[6] A. Wieder and B. B. Brandenburg, "Efficient partitioning of sporadic real-time tasks with shared resources and spin locks," in *2013 8th IEEE International Symposium on Industrial Embedded Systems (SIES)*, June 2013.

[7] A. Biondi, P. Pazzaglia, A. Balsini, and M. Di Natale, "Logical execution time implementation and memory optimization issues in autosar applications for multicores," in *International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)*, 2017.

[8] A. Biondi, G. Buttazzo, and M. Bertogna, "A design flow for supporting component-based software development in multiprocessor real-time systems," *Real-Time Systems*, Oct 2018.

[9] I. Shin and I. Lee, "Periodic resource model for compositional real-time guarantees," in *24th IEEE Real-Time Systems Symposium*, 2003.

[10] A. Davare, Q. Zhu, M. Di Natale, C. Pinello, S. Kanajan, and A. Sangiovanni-Vincentelli, "Period optimization for hard real-time distributed automotive systems," in *2007 44th ACM/IEEE Design Automation Conference*, June 2007.

[11] B. B. Brandenburg, "A fully preemptive multiprocessor semaphore protocol for latency-sensitive real-time applications," in *2013 25th Euromicro Conference on Real-Time Systems*, July 2013, pp. 292–302.

[12] R. Cavicchioli, N. Capodieci, and M. Bertogna, "Memory interference characterization between CPU cores and integrated GPUs in mixed-criticality platforms," in *2017 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, Sep. 2017.

[13] 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)*, Dec 2017.

[14] D. Casini, A. Biondi, G. Nelissen, and G. Buttazzo, "Memory feasibility analysis of parallel tasks running on scratchpad-based architectures," in *2018 IEEE Real-Time Systems Symposium (RTSS)*, Dec 2018.

[15] G. Nelissen, J. Fonseca, G. Raravi, and V. Nélis, "Timing analysis of fixed priority self-suspending sporadic tasks," in *2015 27th Euromicro Conference on Real-Time Systems*, July 2015.

[16] R. Henia, A. Hamann, M. Jersak, R. Racu, K. Richter, and R. Ernst, "System level performance analysis - the SymTA/S approach," *IEEE Proceedings - Computers and Digital Techniques*, March 2005.

[17] J. Rox and R. Ernst, "Compositional performance analysis with improved analysis techniques for obtaining viable end-to-end latencies in distributed embedded systems," *Int. J. Softw. Tools Technol. Transf.*, vol. 15, no. 3.

[18] J. Fonseca, G. Nelissen, V. Nelis, and L. M. Pinho, "Response time analysis of sporadic DAG tasks under partitioned scheduling," in *2016 11th IEEE Symposium on Industrial Embedded Systems (SIES)*, May 2016.

[19] D. Casini, A. Biondi, G. Nelissen, and G. Buttazzo, "Partitioned fixed-priority scheduling of parallel tasks without preemptions," in *2018 IEEE Real-Time Systems Symposium (RTSS)*, Dec 2018.

[20] D. Casini, T. Blaß, I. Lütkebohle, and B. Brandenburg, "Response-time analysis of ROS 2 processing chains under reservation-based scheduling," in *Proceedings 31th Euromicro Conference on Real-Time Systems (ECRTS 2019)*, 2019.