A New Modelling Framework for Coarse-Grained Programmable Architectures

Elias Barbudo, Eva Dokladalova, Thierry Grandpierre

Compas'2020 : Parallélisme/ Architecture/ Système/ Temps Réel
MILC - Lyon, France, du 30 juin au 3 juillet 2020

A New Modelling Framework for Coarse-Grained Programmable Architectures

Elias Barbudo, Eva Dokladalova, Thierry Grandpierre

Université Gustave Eiffel,
Laboratoire d'Informatique Gaspard Monge (LIGM), UMR CNRS 8049, ESIEE Paris
{surname.lastname}@esiee.fr

Abstract
Coarse-grained reconfigurable architectures (CGRA) are designed to deliver high-performance computing while drastically reducing the latency of the computing system. Although they are often highly domain-specifically optimized, they keep several levels of flexibility so that they can be reused. However, their reuse is generally limited due to the complexity of identifying the best allocation of new tasks into the hardware resources. Another limiting point is the complexity to produce a reliable performance analysis for each new implementation.
To solve this problem, we propose to consider CGRA as a programmable, configuration-driven computing fabric, called Coarse-Grained Programmable Architecture (CGPA). We propose a new latency-based model to describe all hardware elements. We demonstrate how to implicitly model, with the help of latency's prediction, the heterogeneity of their material implementations. Our model provides the possibility to assess also the configuration cost, often neglected in other works.
The design of the modelling framework allows it to become a part of a complete application mapping and scheduling chain, up to the automated generation of the execution context, thus maximizing the reusability of the given CGPA.

Keywords : Hyper-graph, coarse-grained programmable architecture, hardware model, latency, performance analysis.

1. Introduction

Today, we observe the acceleration of autonomous systems utilization almost in all branches of industry [1], shipyards (robots) [12], transport (vehicles) [9], and even in construction sites [15]. These systems evolve in uncontrolled conditions. To interact with their environment, they need to perform high-performance computations under hard real-time constraints with high reactivity, expressed as low latency processing of sensors information.
To answer these constraints, an extensive number of hardware architectures have been proposed in the past, trying to find the best trade-off between performance and flexibility. We can cite some examples going from arrays of General-Purpose Processors (GPPs), passing through Networks on Chip (NoCs), Field Programmable Gate Arrays (FPGAs) solutions up to Coarse-Grained Reconfigurable Architectures (CGRAs) [8]. In the above-mentioned context, the latter ones provide the best ratio between the increase of the overall performance while decreasing computing latency and minimizing the energy budget [13].
CGRAs are high-performance platforms optimized for a given application domain. They consist of sets of processing, communication and memory resources (Fig. 1).

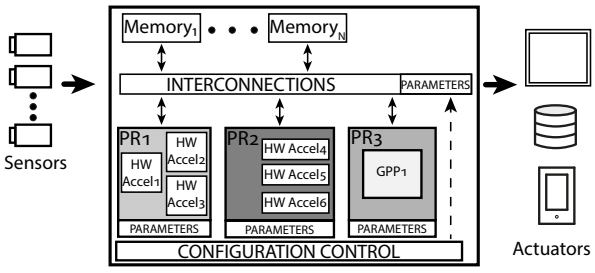

FIGURE 1 – Global architecture of a CGPA.

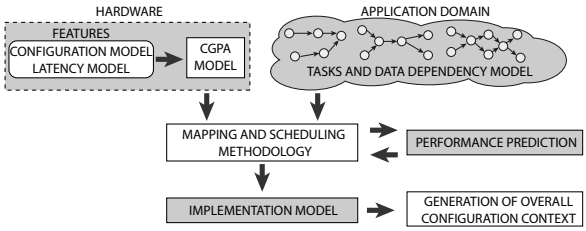

FIGURE 2 – Overall scheme of general framework principles for CGRA/CGPA efficient reuse.

The processing resources are heterogeneous modules, with possible different computing models designed to perform a specific set of tasks. Each processing has a set of programmable parameters. Notice that the set of parameters of each processing resource can be different. The communication resources allow creating adequate datapaths between processing resources.

Also, a CGRA tends to process the applications in a dense pipeline. It means, as soon as some processed result is available, it is immediately communicated to the next processing resource according to the application data chain. Considering the heterogeneous computing models combined with the cost of memorization, communication, and configuration, the efficient reuse of the hardware is not trivial. The solution is an automated mapping and scheduling tool (Fig. 2). The core of this tool must be a model allowing to describe the heterogeneity of programming models of any CGRA hardware element.

The advances in the field of hardware modelling are significant, but they do not fulfill the needs of CGRA modelling. The current models do not provide the necessary means to describe latency at a cycle-accurate level [3, 4] or neglect some resources to reduce the complexity of the performance analysis [10, 2]. Also, considering the need to optimize the reactivity of the systems, the tool has to provide accurate, near to real performance latency estimation. Several works propose methods to compute latency [14, 11, 5, 7, 6], however, these methods often neglect the configuration cost of the system [2] and in general, they provide pessimistic values.

In this paper, we present a new accurate model of Coarse-Grained Programmable Architectures (CGPA), based on the modelization of resource heterogeneity through fine latency evaluation. In our work, we decide to use the term "programmable" instead of "reconfigurable" architectures. The main contributions of this work are :

— A new formal model of CGPA, based on hyper-graphs. Our model covers all the hardware resources, including (re-)configuration management, memory, communication, and processing resources.
— Cycle-accurate latency performance analysis, without neglecting configuration cost of any part of CGPA.

This model can be easily integrated into a complete mapping framework and provide the required means of reusability for the CGPA. Fig. 2 shows the modules of a framework for CGPA efficient reuse, this paper covers the modules in gray.

The organization of the remaining part of this paper is as follows. Section 2 introduces the new model of CGPA. In Section 3, we introduce a latency-based performance analysis. Then Section 4 describes the experimental study. Finally, Section 5 summarizes this paper and outlines the perspectives of this work.

2. Latency Based Models

In this section, we describe three models. We consider the application and hardware model as inputs of a mapping algorithm, and the implementation model the output.

2.1. Application model

Let $G_{APP}(T, D)$ be a directed hyper-graph that models an application. T is a set of nodes that represents the tasks of the application. D is a set of oriented hyper-edges that represents data dependency between tasks. We call a task $t_i \in T$, so that $t_i = (type_i, p_i)$, where $type_i$ is the type of transformation applied to the data and p_i is the set of the transformation parameters.

2.2. CGPA hardware model

In our model, a directed hyper-graph $G_{HW}(S, K)$ represents a CGPA architecture, where the set of nodes (S) represents the hardware resources and the set of oriented hyper-edges (K) models the hardware resources interconnections. Fig. 3 shows the general hierarchy of the subsets of S. We detail these subsets in the following sections.

2.2.1. Sequencer node s^{cfg}

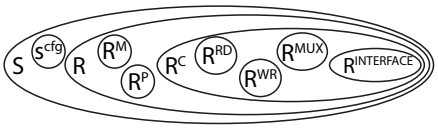

FIGURE 3 – General hierarchy of the hardware resources (S).

The sequencer node s^{cfg} controls the system configurations. It is in charge of the modification of the resources configuration between different applications or between partial configurations required to realize one application. We define $s^{cfg} = (Cfg^{fun}, Cfg^{param})$, where Cfg^{fun} is a set of designer-defined functions allowing to express how to compute the configuration cost of each hardware resource according to the implemented configuration mechanism. Finally, Cfg^{param} is the set of configuration parameters of the hardware resources.

2.2.2. Hardware resources R

R is the set of hardware resources dedicated to transform (process) R^P, store R^M or communicate data R^C.

Processing resources R^P

The subset R^P represents resources that apply a given transformation of the input data. We define $r_i^P \in R^P$, $r_i^P = (\mathcal{T}_i, \Pi_i, \mathcal{L}_i, Cfg_i)$, where $\mathcal{T}_i$ is the set of tasks that r_i^P can perform and Π_i is the set of allowed parameters of each task. $\mathcal{L}_i = (\mathcal{L}_i^{IN}(\mathcal{T}_i, \Pi_i), \mathcal{L}_i^{CL}(\mathcal{T}_i, \Pi_i))$ represents a tuple of functions assigning the input and computing latencies values of r_i^P, depending on $\mathcal{T}_i$ and Π_i as input parameters. To express the exact execution time of a task on any hardware resource, we propose using its **input latency** and **computing latency**. We define ***input latency*** as the number of clock cycles necessary to read all the samples required to start to compute the first result. We define ***computing latency*** as the number of clock cycles necessary to produce the result once all input samples are available.

The parameter $Cfg_i \in Cfg^{fun}$ defines the configuration cost of r_i^P. Cfg_i=$Cfg_i(\mathcal{T}_i, \Pi_i)$ is again a designer-defined function having the possibility to assign the configuration cost value of r_i^P. To complete, each r_i^P is able to implement the tasks *copy* and *disable* allowing to manage correctly unused processing resources in the data-path.

Communication resources R^C

The subset R^C represents the resources dedicated to the data transfer, copy and data-path control. A multiplexer $r_i^{MUX} \in R^{MUX}$ provides a set of inputs and outputs, and performs a copy operation from a selected input to the selected output. An r^{MUX} can describe a mux/demux/ arbitrary/switch and is model as a four element tuple $(I_i^{port}, O_i^{port}, \mathcal{L}_i, Cfg_i)$. I_i^{port} and O_i^{port}, are the set of input and output ports of r_i^{MUX}. $\mathcal{L}_i$ is the latency of r_i^{MUX} and $Cfg_i \in Cfg^{fun}$ represents the configuration cost. An $r_i^{WR} \in R^{WR}$ and an $r_i^{RD} \in R^{RD}$ are resources able to perform a *write/read* operation from or to a memory resource. We define r_i^{WR} and r_i^{RD} with a three-

element tuple $(\mathcal{A}_i, \mathcal{L}_i, Cfg_i)$. $\mathcal{A}_i$ defines the address space to access. $\mathcal{L}_i$ models the latency of the *write/read* operation and $Cfg_i \in Cfg^{fun}$ represents the possible configuration cost. The external sources and consumers of the data are $r_i^{sensor}, r_i^{actuator} \in R^{interface}$. We describe an r_i^{sensor} with a tuple $(\Pi_i, \mathcal{L}_i^{sensor})$, where Π_i are the allowed parameters and $\mathcal{L}_i^{sensor}$ is the latency of producing one data sample. We consider that an r_i^{sensor} has an internal r^{WR}, which allows it to transfer data directly to an r^P or an r^{MUX}, or write data to an r^M. An $r_i^{actuator}$ is also described in the same manner.

Memory resources R^M

The subset R^M represents the hardware memory resources (RAM modules, sequential memory modules). We describe each $r_i^M \in R^M$ with a tuple $(\mathcal{A}_i, C_i^{RD}, C_i^{WR})$, where $\mathcal{A}_i$ represents the addressing space of r_i^M. C_i^{RD} is the number of read channels available and C_i^{WR} the number of write channels. Notice that the memory resources do not have the expression of the latency, this one is always integrated into the associated r^{RD} and r^{WR} nodes.

2.3. Implementation model

We consider $G_{MAP}(S', K')$ as the output of a mapping algorithm. G_{MAP} contains all the fixed parameters obtained by mapping G_{APP} onto G_{HW}.

We describe $G_{MAP}(S', K')$ as a directed weighted hyper-graph with fixed parameters for each resource. $K' = K'_1, K'_2, \ldots, K'_m$ represents a set of oriented weighted hyper-edges, where the weight of each hyper-edge is equal to $l_i = \mathcal{L}_i(\tau_i, \pi_i)$ of the head node. Several different instances $G_{MAPi}(S', K')$ (called *time slot*) may construct the implementation graph. A time slot is a subset of hardware resources configured to perform a subset of application tasks if one or more reconfigurations are needed to finish the complete application. In other words, we define a time slot as the interval from the resources configuration stage until the last result is outputted at the end of the configured data pipeline.

We define $S' = R' \cup S^{cfg}$, where S^{cfg} is a set of ordered nodes that represents the configuration control of each time slot. The sequencer node s^{cfg} is in charge to generate S^{cfg}. There should be the same number of $s'^{cfg} \in S^{cfg}$ as the number of time slots. Recall that R' is the set of hardware resources with fixed parameters. We consider $R' = R^p \cup R^c \cup R^m \cup R^{sn}$. For the subsets, R^p, R^c and R^m the descriptors are similar to the hardware model with the only difference of the fix parameters.

2.3.1. Time slot dependency node set R^{sn}

This is an artificial subset of nodes representing data dependency between time slots. If an application can not be fully implemented in one time slot, we need to store the processed data at the end of the time slot i and read it in time slot $i+1$. We introduce the notion of the special node $r^{sn} \in R^{sn}$, which will connect the two r^m and allow us to identify the data dependency.

3. Latency-based performance analysis

Computing cost (CC) is a latency-based metric that is fundamental for critical time systems development as they must comply with hard real-time constraints. In this context, we compute CC as follows :

$$CC = \sum_{i=1}^{N} (Tc_i + \overbrace{(CL_i)(Height)(Width)}^{TEX_i} + \overbrace{\sum_{j=1}^{|CP_i|-1} ((\mathcal{L}_j^{IN} - 1)(\alpha_j) + \mathcal{L}_j^{CL} + 1))}^{TIN_i} \quad (1)$$

Where N is the number of time slots of the implementation graph, TIN_i is the overall input latency of time slot i, TEX_i is the execution duration of time slot i and Tc_i is the configuration

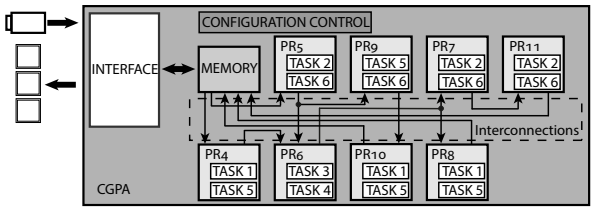

FIGURE 4 – Architecture organization of the study case.

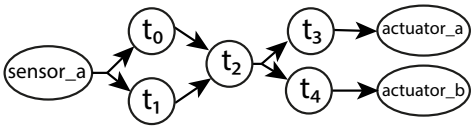

FIGURE 5 – Application model of the example.

cost of time slot i. CP_i is a set of resources that belong to the critical path of time slot i. $\mathcal{L}_j^{IN}$ is the input latency of the resource, $\mathcal{L}_j^{CL}$ is the computing latency of the resource. Finally, α_j is an expression of the propagation of computing latency. Let $\alpha_j = max(\alpha_{j-1}, \mathcal{L}_{j-1}^{CL})$, where α_{j-1} is the α of the predecessor and $\mathcal{L}_{j-1}^{CL}$ is the computing latency of the predecessor. CL_i is the worst computing latency of the critical path of time slot i, *Height* and *Width* are the height and width of the input image (resolution of the image).

4. Experimental study

In this section, we present an experimental study. We use three latencies configurations to show the reliability of our model and performance analysis. Consider the hypothetical CGPA example in Fig. 4. It consists of 8 image processing resources (PR_i) and a memory block. It has one sensor as a producer of data and three actuators as consumers. In Fig. 6 we can see the hardware model. The second input of our experimental study is the image processing application showed in Fig. 5. It consists of five image processing operators as tasks.

4.1. Experimental settings

Hardware parameters. Table 1 lists the type of tasks ($\mathcal{T}_i$) that each processing resource can implement. For this example, we consider Π_i as fixed for all the processing resources. Consider input latency as 2 samples and computing latency as 2 clock cycles for all the processing resources. This is the first set of latency features.

TABLE 1 – Experimental study : processing resources features.

	$\mathcal{T}_i$
$r^P_{4,8,10}$	task1, task5
$r^P_{5,7,11}$	task2, task6
r^P_9	task5, task6
r^P_6	task3, task4

Application parameters. The parameters of the tasks are $t_0 = (task1)$, $t_1 = (task2)$, $t_2 = (task3)$, $t_3 = (task2)$, $t_4 = (task1)$. In this example p_i is fix.

4.2. Implementation model

Applying manual mapping we get the implementation model showed in Fig. 7, which only needs one time slot. Notice that the sequencer node s^{cfg} generates the configuration node s'^{cfg}.

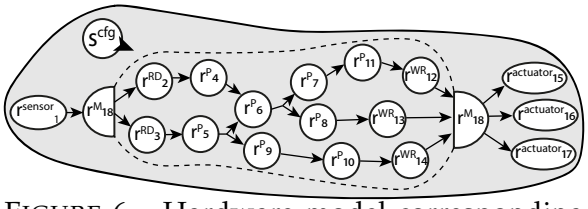

FIGURE 6 – Hardware model corresponding the case study CGPA.

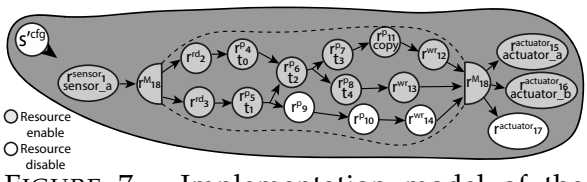

FIGURE 7 – Implementation model of the example.

4.3. Performance analysis

We continue with the performance analysis using the first set of latency features. We can see the final timing diagram of the implementation in Fig. 8.

Notice that the maximum configuration cost corresponds to the path showed in Fig. 10. We compute the performance analysis over this path. Consider an input image resolution of 100x100 pixels. Consider the configuration cost of the time slot is equal to one clock cycle. According to Equation 1, the computing cost is 20019 clock cycles per image.

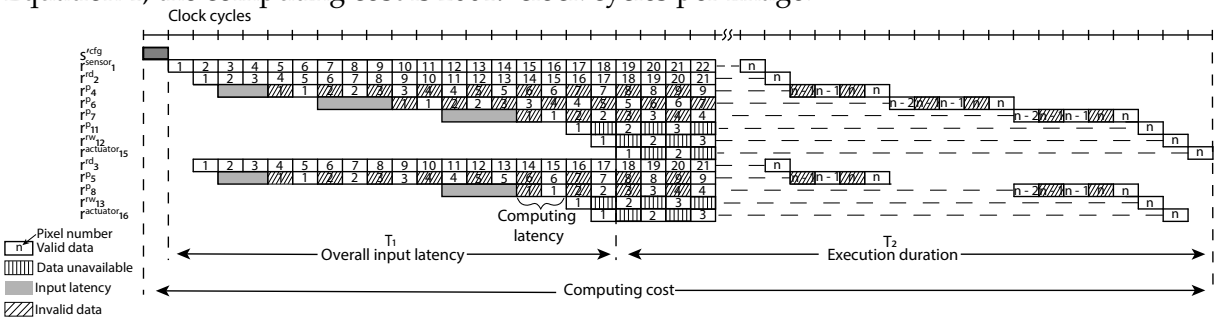

FIGURE 8 – Timing diagram of the implementation graph under the first set of latency features.

Now, let's use a different set of latency features. We change the parameters for r_5^P. Let's assume that for the implementation of *task2*, the input latency changes to 3 samples and the computing latency to 3 clock cycles. The critical path is no longer the same. Fig. 11 shows the new critical path and the resulting computing cost is 30023 clock cycles.

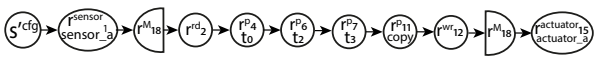

FIGURE 10 – Critical path with the first latency features.

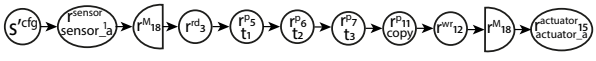

FIGURE 11 – Critical path with the second latency features.

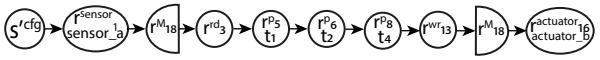

FIGURE 9 – Critical path with the third latency features.

Finally, for the third considered latency features, the changes are considerable. For r_5^P, the implementation of `task2` considers 3 samples as input latency and 2 clocks cycles as computing latency. For r_6^P, the implementation of `task3` considers 4 samples as input latency and 3 clocks cycles as computing latency. For r_8^P, the implementation of `task1` considers 1 sample as input latency and 3 clocks cycles as computing latency. We get a new critical path (Fig. 9) and the computing cost is 30025 clock cycles. The performance analysis is able to identify the slightest change in the latencies of the resources, either due to different material implementations or because of changes in the parameters.

5. Conclusions

In this paper, we presented a latency based model for CGPA. We show the properties of our approach, and prove that for latency analysis is precise. Due to the characteristics of the model, we pretend to extend to other types of hardware accelerators. As latency is a crucial factor for autonomous systems, this model is ideal for the development of new systems and can predict the behavior of it. We also presented a new performance analysis equation. It takes into account the propagation of computing latency through the processing pipeline. It can provide cycle-accurate results and be a tool for the measurement of the configuration cost of an implementation in a design space exploration. The future work consists in to develop a mapping algorithm for CGPA based on the presented model and integrate the entire set of tools in a complete mapping framework.

References

1. Aazam (M.), Zeadally (S.) et Harras (K. A.). – Deploying fog computing in industrial internet of things and industry 4.0. *IEEE Transactions on Industrial Informatics*, vol. 14, n10, Oct 2018, pp. 4674–4682.
2. Barbudo (E.), Dokládalová (E.), Grandpierre (T.) et George (L.). – A new mapping methodology for coarse-grained programmable systolic architectures. – In Stuijk (S.) (édité par), *SCOPES '19, Sankt Goar, Germany, May 27-28, 2019*, pp. 5–12. ACM, 2019.
3. Bingfeng Mei, Vernalde (S.), Verkest (D.), De Man (H.) et Lauwereins (R.). – Dresc : a retargetable compiler for coarse-grained reconfigurable architectures. – In *2002 IEEE International Conference on Field-Programmable Technology, 2002. (FPT). Proceedings.*, pp. 166–173, Dec 2002.
4. Chin (S. A.), Sakamoto (N.), Rui (A.), Zhao (J.), Kim (J. H.), Hara-Azumi (Y.) et Anderson (J.). – Cgra-me : A unified framework for cgra modelling and exploration. – In *2017 IEEE 28th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, pp. 184–189, July 2017.
5. Feiertag (N.), Richter (K.), Nordlander (J.) et Jönsson (J. Å.). – A compositional framework for end-to-end path delay calculation of automotive systems under different path semantics. – In *RTSS 2009*, 2008.
6. Kloda (T.), Bertout (A.) et Sorel (Y.). – Latency analysis for data chains of real-time periodic tasks. – In *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*volume 1, pp. 360–367, Sep. 2018.
7. Krawczyk (L.), Bazzal (M.), Govindarajan (R. P.) et Wolff (C.). – Model-based timing analysis and deployment optimization for heterogeneous multi-core systems using eclipse app4mc. – In *ACM/IEEE 22nd International Conf. on Model Driven Engineering Languages and Systems Companion (MODELS-C)*, pp. 44–53, Sep. 2019.
8. Liu (L.), Zhu (J.), Li (Z.), Lu (Y.), Deng (Y.), Han (J.), Yin (S.) et Wei (S.). – A survey of coarse-grained reconfigurable architecture and design : Taxonomy, challenges, and applications. *ACM Comput. Surv.*, vol. 52, 2019, pp. 118 :1–118 :39.
9. Naufal (J. K.), Camargo (J. B.), Vismari (L. F.), de Almeida (J. R.), Molina (C.), González (R. I. R.), Inam (R.) et Fersman (E.). – A2cps : A vehicle-centric safety conceptual framework for autonomous transport systems. *IEEE Transactions on Intelligent Transportation Systems*, vol. 19, n6, June 2018, pp. 1925–1939.
10. Pelcat (M.), Desnos (K.), Maggiani (L.), Liu (Y.), Heulot (J.), Nezan (J.) et Bhattacharyya (S. S.). – Models of architecture : Reproducible efficiency evaluation for signal processing systems. – In *2016 IEEE International Workshop on Signal Processing Systems (SiPS)*, pp. 121–126, Oct 2016.
11. Rajeev (A. C.), Mohalik (S.), Dixit (M. G.), Chokshi (D. B.) et Ramesh (S.). – Schedulability and end-to-end latency in distributed ecu networks : Formal modeling and precise estimation. – In *Proceedings of the Tenth ACM International Conference on Embedded Software, EMSOFT '10*, EMSOFT '10, p. 129–138, 2010.
12. Wang (R.), Wei (Y.), Song (H.), Jiang (Y.), Guan (Y.), Song (X.) et Li (X.). – From offline towards real-time verification for robot systems. *IEEE Transactions on Industrial Informatics*, vol. 14, n4, April 2018, pp. 1712–1721.
13. Wijtvliet (M.), Waeijen (L.) et Corporaal (H.). – Coarse grained reconfigurable architectures in the past 25 years : Overview and classification. – In *International Conference on Embedded Computer Systems : Architectures, Modeling and Simulation (SAMOS)*, pp. 235–244, July 2016.
14. Wilhelm (R.), Engblom (J.), Ermedahl (A.), Holsti (N.), Thesing (S.), Whalley (D.), Bernat

(G.), Ferdinand (C.), Heckmann (R.), Mitra (T.) et al. – The worst-case execution-time problem—overview of methods and survey of tools. *ACM Trans. Embed. Comput. Syst.*, vol. 7, n 3, mai 2008.

15. Yan (R.), Kayacan (E.), Chen (I.), Tiong (L. K.) et Wu (J.). – Quicabot : Quality inspection and assessment robot. *IEEE Transactions on Automation Science and Engineering*, vol. 16, n2, April 2019, pp. 506–517.