Response time analysis of parameterized dataflow applications on heterogeneous SW/HW systems

Claudio Rubattu

▶ **To cite this version:**

Claudio Rubattu. Response time analysis of parameterized dataflow applications on heterogeneous SW/HW systems. Embedded Systems. INSA de Rennes, 2020. English. NNT : 2020ISAR0005 . tel-03157774

HAL Id: tel-03157774
https://theses.hal.science/tel-03157774v1

Submitted on 3 Mar 2021

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

DOCTORAT BRETAGNE LOIRE MATHSTIC

INSA RENNES

THÈSE DE DOCTORAT DE

L'INSTITUT NATIONAL DES SCIENCES APPLIQUEES RENNES

ÉCOLE DOCTORALE N° 601
Mathématiques et Sciences et Technologies de l'Information et de la Communication
Spécialité : *Signal, Image, Video*

Par

« Claudio RUBATTU »

« Response time analysis of parameterized applications on heterogeneous HW/SW systems »

Thèse présentée et soutenue à Rennes, le 02.12.2020
Unité de recherche : IETR, UMR CNRS 6164
Thèse N° : 20ISAR 22 / D20 - 22

Rapporteurs avant soutenance :

Abdoulaye Gamatié	Directeur de Recherche CNRS - LIRMM - UMR 5506 CNRS & UM (France)
Stéphane Mancini	Maître de conférences - Laboratoire TIMA / CDSI (France)

Composition du Jury :

Président :	Luigi Raffo	Professeur - Università degli Studi di Cagliari (Italie)
Examinateurs :	Shuvra S. Bhattacharyya	Professeur - University of Maryland (USA)
	Jeronimo Castrillon	Professeur - Technische Universität Dresden (Allemagne)
	Eduardo Juárez Martínez	Professeur Agrégé - Universidad Politécnica de Madrid (Espagne)
	Francesca Palumbo	Professeur Agrégé - Università degli Studi di Sassari (Italie)
Dir. de thèse :	Maxime Pelcat	Maître de conférences - IETR UMR CNRS 6164 (France)

DOCTORAT BRETAGNE LOIRE MATHSTIC

INSA RENNES

Response time analysis of parameterized applications on heterogeneous HW/SW systems

Claudio RUBATTU

Document protégé par les droits d'auteur

Table of Contents

Acknowledgements

I want to thank my advisors Dr Francesca Palumbo and Dr Maxime Pelcat for making this experience possible and for their support during these three years. Francesca, thank you for motivating me to achieve my goals and for letting me explore various aspects related to research. Maxime, thank you for welcoming me at INSA and providing me with motivating research insights.

I am grateful to Dr Abdoulaye Gamatie and Dr Stephane Mancini for reviewing this thesis. Thanks to Pr Luigi Raffo for presiding the jury and to Pr Shuvra S. Bhattacharyya, Pr Jeronimo Castrillon and Dr Eduardo Juárez for being members of the jury. Thanks Pr Jean-Francois Nezan, of the VAADER team from IETR, and Dr Paolo Meloni, of the EOLAB group from the University of Cagliari, for being part of my PhD committee.

It has been a pleasure to work with the members of the VAADER team from IETR. Thanks to Dr Karol Desnos, Dr Antoine Morvan and Florian Arrestier for their technical help and computing insights. Thanks to Alexandre Honorat for his technical support, friendship, and help during my visiting periods in Rennes. Thanks to Thomas Amestoy for making it easier for me to socialize in the team. Thanks to Frederic Garesche for his IT support and to Corinne Calo and Aurore Gouin for their administrative support.

A special thanks to the members of the IDEA group from the University of Sassari and of the EOLAB group from the University of Cagliari. Thanks to Dr Luca Pulina and Simone Vuotto for sharing technical insights in computing science and their friendship. Thanks to Dr Carlo Sau for his technical support and the unforgettable moments in Montenegro. Thanks to Dr Tiziana Fanni for her co-working effort and friendship. Thanks to Dr Laura Pandolfo for her advices and moments of fun. Thanks to Monica Marini and Maria Grazia Sini for their administrative support and friendship.

Many thanks to the members of the CEI and CITSEM centers from Universidad Politécnica de Madrid. Thanks to Dr Eduardo de la Torre, Dr Alfonso Rodríguez, and Daniel Madroñal for their support and friendship. Thanks to Leonardo Suriano for sharing technical knowledge, his friendship and incredible moments. Thanks to Dr Rubén Salvador for his motivation, friendship and moral support in longer working evenings.

Finally, I am deeply grateful to my family and friends for their help and support.

Abstract

In contexts such as embedded and cyber-physical systems, the design of a desired functionality under constraints increasingly requires a parallel execution of different tasks on heterogeneous architectures. The nature of such parallel systems implies a huge complexity in understanding and predicting performance in terms of response time. Indeed, response time depends on many factors associated with the characteristics of both the functionality and the target architecture.

State-of-the art strategies derive response time by examining the operations required by each task for both processing and accessing shared resources. This procedure is often followed by the addition or elimination of potential interferences due to task concurrency. However, such approaches require an advanced knowledge of the software and hardware details, rarely available in practice.

This thesis provides an alternative "top-down" strategy aimed at extending the cases in which hardware and software response times can be analyzed and predicted. The proposed strategy leverages on dataflow-based application representations and focuses on the response time estimation of reconfigurable applications mapped on both general-purpose and specialized processing elements.

Chapter 1

Introduction

Nowadays, embedded systems are certainly the most common devices in mass and industrial electronics. Multiprocessor System on Chip (SoC)s (MPSoCs), on which are handled their processing phases, embed a growing number of heterogeneous Processing Elements (PEs) in order to efficiently perform modern functionalities. For this reason, the design of systems based on MPSoCs has become increasingly complex and several approaches aiming at limiting this complexity have appeared [CF16]. Among the concepts of embedded systems, Cyber-Physical Systems (CPSs) have become known and studied with interest by the scientific community in the last few years. These systems are capable of monitoring and controlling physical elements and consider heterogeneous components that interact with each other in different modalities depending on the context in which they operate [Raj+10]. Design and maintenance of such systems are extremely complex because of their multidisciplinary nature, their elaborate requirements, the heterogeneity of their components and the continuous communication between their physical and cyber layers [DLS12]. Both embedded systems and CPSs imply demanding requirements in terms of flexibility and efficiency. The former property requires solutions capable of performing different functionalities evaluated in various operating modes. The latter is needed for the exploitation of the system with respect to the performance and costs deriving from the design choices (e.g. regarding requirements associated with energy budget and response time).

In order to achieve flexibility and efficiency, there is a need for digital hardware and software capable of implementing different functionalities with multiple operating modes in order to meet the requirements. An increasingly considered solution is represented by the combination of reconfigurable implementations both in terms of hardware and software [Sha+14; GPK19; Pér+20]. However, in the presence of multiple operating modes that depend on the state of the system and the external environment, reconfiguration may not be sufficient to guarantee complete flexibility and/or an appropriate degree of efficiency. Indeed, in cases such as the failure of system components or the need to

change functionality, a runtime management of the execution is required. Nevertheless, adaptation according to uncertain events and to changing functional and non-functional requirements is an important challenge for system developers [LCK16]. In order to facilitate reconfiguration and application of adaptation strategies, the use of parameters associated with specific operating modes offers high-level management of functionality. Since these operating modes can concern both hardware (HW) and software (SW) implementations, parametric functionalities performed on heterogeneous systems, aimed at providing flexibility and efficiency, are increasingly evaluated.

In this context, CERBERO [1] H2020 and FitOptiVis [2] ECSEL European projects aim at developing design environments for CPSs. CERBERO is based on two main elements: i) a cross-layer and model-based approach to describe, optimize and analyze the system according to different views; and ii) an extended adaptation of the calculation to the system state as well as to its environment, provided by an autonomous adaptation engine. FitOptiVis is focused on a cross-domain approach aiming at achieving multi-objective optimization for performance and energy use, which is aimed at adapting distributed image and video processing pipelines of CPSs. This thesis is mainly focused on the second CERBERO objective, which also applies to FitOptiVis, and, in particular, on the study of design methodologies in order to apply reconfiguration and adaptation of parametric functionalities executed on heterogeneous systems.

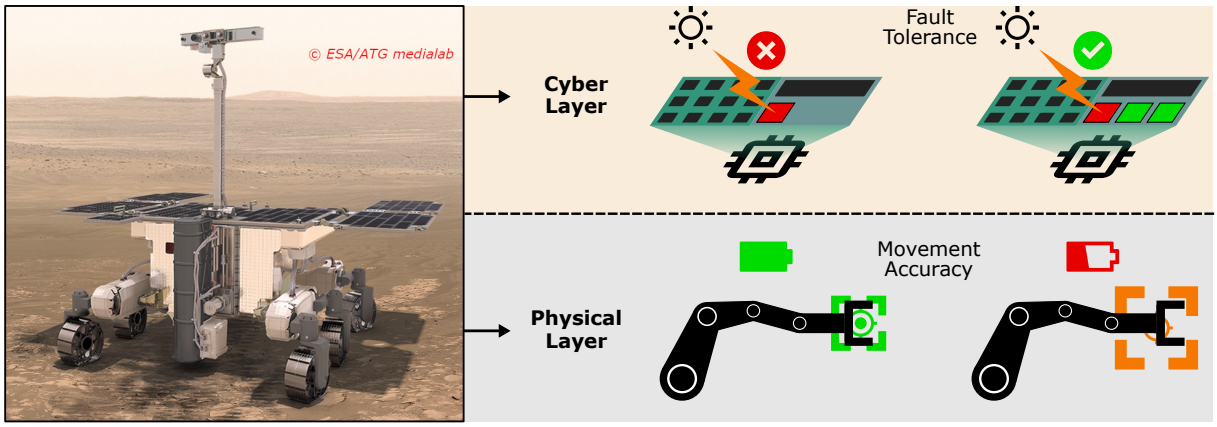


Figure 1.1 – Example of adaptation in a CPS, based on a use case of the CERBERO H2020 European Project.

As an example of adaptation in CPS, Figure 1.1 depicts a use case of CERBERO

1. `https://www.cerbero-h2020.eu`
2. `https://fitoptivis.eu`

(provided by: Thales Alenia Space), which consists of a self-healing system for planetary exploration aiming at moving across the surface and studying future planet at depth. The cyber layer considers a heterogeneous HW architecture, which can be reconfigured with multiple PEs aimed at performing the same processing in order to provide fault tolerance through output data redundancy, since the effect of solar radiation may produce a malfunction of the electronics on board. The cyber layer adapts the system based on information from the physical layer to control its physical components, e.g. moving a robotic arm with a certain degree of accuracy depending on the current level of energy stored in the battery.

Among the most common Key Performance Indicators (KPIs) of embedded systems and CPSs, response time plays a relevant role for the strategies considered for reconfiguration and adaptation of signal processing solutions [HAR14; GPM14]. Indeed, this metric represents the execution time of a complete elaboration of a functionality, starting from the acquisition of a given input data and ending with the generation of an associated output data. Nevertheless, the accurate evaluation of the response time implies complex timing analyses and/or strong limitations in the implementation of the HW and SW components. Indeed, although different approaches in literature aim at providing accuracy and reliability in the verification of the response time, their applicability depends on various aspects, such as: the availability of information regarding HW and SW details, and the amount of processing and requests in the use of system resources associated with the functionality [Wil+08]. In addition, in contexts demanding reconfiguration and adaptation, on-line timing evaluation and budgeting may require complex models execution, not compatible with applicative performance requirements. Therefore, this issue remains far from being solved effectively for functionalities that require flexibility and efficiency in their execution. For this reason, there is a need to identify an alternative approach in order to simplify the response time estimation, especially when parametric functionalities are performed on heterogeneous systems. However, reducing complexity does not come for free, and typically is linked to a loss of response time evaluation accuracy. The analysis must therefore maintain the robustness of the estimated values, presenting at least a high degree of fidelity with respect to the trend of the real cost of the functionality in terms of response times.

A promising approach capable of offering a new perspective with respect to the state of the art consists in characterizing a functionality as its *activity* in terms of a chosen metric, and then evaluating its associated cost when mapped upon an Model of Architec-

ture (MoA) [Pel+18]. Such a strategy leverages on a high level of abstraction, which is a fundamental prerequisite to allow a reduction of complexity. In addition, MoAs can be designed and taylored in order to obtain linearity in the relation between cost and activity. In particular, an approach based on linear MoAs has been proved to have a high degree of fidelity for the energy estimation of parametric functionalities mapped on heterogeneous systems. Nevertheless, MoA-based approach has not yet been applied in the context of estimating timing metrics. In addition to the purpose of reducing model complexity, such a strategy may also favor the elasticity of the timing analysis, since the activity can be selected depending on the factors that the developer considers crucial in determining response time. In fact, elastic analyses are convenient when applying reconfiguration and adaptation of parametric functionalities on heterogeneous MPSoCs [Pim+02; Hen+11]. In particular, since such systems must satisfy flexibility and efficiency requirements, they need to be capable of handling generic and customized specifications of parametric functionalities. Therefore, a combination of general-purpose PEs and specific hardware with low reconfiguration overhead may represent a proper trade-off in order to provide flexibility and efficiency at the same time. Nevertheless, design and management of MPSoCs composed of such elements leads to consider different development methods, which imply specific levels of knowledge and estimation opportunities. For these reasons, the use of tools with demonstrated capabilities in terms of reconfiguration and adaptation of such components is required. Given the H2020 CERBERO context and the complementarity of the tools built within the CERBERO consortium, a decision has been taken early in the project to concentrate on the tools described in Section 2.4 rather than integrating external tools into the process. Within the CERBERO project, a design flow evaluating the tools MDC and SPIDER, which present common properties of modularity, parametrization and availability of their source code, can provide proper solutions to be implemented in order to provide reconfiguration and adaptation of such combined systems. Indeed, while MDC manages hardware datapath reconfiguration at runtime, SPIDER handles the adaptation of a parallel software datapath. However, both tools do not integrate response time analysis methods among their design features, motivating for the current study of a lightweight response time model.

Objectives of the Thesis

The main objective of this thesis is to offer an elastic approach for response time estimation that aims at supporting adaptation strategies in contexts where parametric

functionalities are implemented with HW and SW dataflow specifications and executed in general-purpose and customized reconfigurable PEs. For this purpose, the following sub objectives have been envisioned:

— Objective 1: Design an alternative approach for the estimation of the response time in the design of customized CG Reconfigurable (CGR) architectures.

— Objective 2: Compose a toolchain for the design and runtime management of functionalities mapped upon systems composed of general-purpose and CGR PEs.

— Objective 3: Imagine a methodology for detecting application activity determining response time associated with parametric functionalities.

— Objective 4: evaluating the linear MoA (called LSLA) with the previously defined activity for the response time estimation, in order to provide a motivational starting point for supporting reconfiguration and adaptation based on simple and generalizable activity of parametric functionalities executed on heterogeneous MPSoCs.

Structure of the Thesis

The thesis manuscript firstly exposes an overview of the research areas at the background of the proposed approach in Chapter 2: dataflow (DF) Models of Computation, CGR acceleration, and timing analysis. In Chapter 3, problem definition, motivation and objectives for each contribution are described. The subsequent three chapters, associated with the contributions of the thesis, are organized as follows:

— Chapter 4 - Contribution 1: a design flow capable of estimating the response time in the design of CGR accelerators (Objective 1) and a framework for the development of parametric applications mapped on CGR and general-purpose PEs (Objective 2) are presented and evaluated.

— Chapter 5 - Contribution 2: a methodology for detecting the determining factors of response time in the execution of parametric functionalities mapped on heterogeneous MPSoCs (Objective 3) is proposed.

— Chapter 6 - Contribution 3: a design flow for estimating system execution latency through the linear MoA, depending on the activity defined in Contribution 2 (Objective 4), is described.

Final considerations related to the thesis work and possible future improvements conclude the manuscript (Chapter 7).

Chapter 2

HW/SW Design of Reconfigurable Systems for Signal Processing Applications

Since this thesis work is focused on estimating the response time of parametric functionalities performed on systems composed of general-purpose and CGR PEs, this chapter provides an overview of the research areas related to this context. Indeed, the evaluation of such KPI depends on the strategies used in the design of the functionality. In particular, such a design can combine specific approaches associated with the development of SW and HW implementations of the functionality. While SW provides versatile processing with arbitrarily complex control paths, a HW implementation tailors processing to the task at hand, reaching higher performance and energy efficiency. From the SW perspective, each description of the functionality is aimed at providing the instructions to be executed in the HW architecture. However, depending on the application model and the target platform considered in the design, timing analysis varies significantly in terms of complexity and accuracy. In fact, the behavior of an application can be described through a combination of operational elements present in a specific set corresponding to a so-called Model of Computation (MoC) [LX04]. Such set and the rules of interaction among its elements represent the semantics of the MoC. Depending on the semantics, a specific strategy for evaluating response time needs to be determined. A MoC aims at formalizing the representations of applications. So as to foster real interoperability and make transformations and exploitation repeatable. On the other hand, the impact of the target architecture on timing is commonly known, since this can have different types and numbers of hardware components. In particular, in addition to an improvement in performance, the implementation of a customized architecture capable of performing only desired operating modes of a functionality or part of this, may lead to more predictable timing behaviors. In general, since describing functionalities through specifications based

on DF MoCs presents properties such as abstraction and modularity which favor desired mapping of its various components, this has been widely used in the evaluation of the response time. For this reason, particular attention will be given to these MoCs and to the use of architectures composed of general-purpose and customized PEs.

This chapter is structured as follows. Section 2.1 presents a focus on the state of the art related to the design of signal processing applications by using DF MoCs. In Section 2.2, an overview of the current hardware (HW) platforms and a detailed description about CGR acceleration and design is proposed. Section 2.3 is focused on the main approaches used in estimating response time. In Section 2.4, tools involved in the design flows proposed as contributions of this thesis have been described. Section 2.5 concludes the chapter.

2.1 Design of Signal Processing Applications

We call a signal processing application a digital processing functionality to be performed on constantly arriving digital data. The model used to describe such a functionality determines the response time estimation strategy. Since DF models present characteristics of modularity and abstraction, these are widely used in predicting application response time. In Section 2.1.1, a brief description of the common systems simulation methods is proposed. Sections 2.1.2 and 2.1.3 respectively report a focus on the models and timing-based tools present in the DF approach.

2.1.1 Overview of the Main System Simulation Methods

In order to describe a functionality, different application models can be applied. This led to the development of many tools capable to also evaluate the functionality in terms of timing behavior [AS19; Men+17; Objb; ABA16]. Different Design Space Exploration (DSE) strategies are based on specific MoCs to study application KPIs [AA18]. However, DSE methods can be grouped into the four families below, on the basis of their application models.

Imperative code simulation (Imp)

The latency of an imperative code with parallel tasks can be predicted before actual execution using an Instruction Set Simulator (ISS) [AS19]. The ISS reproduces the behavior of HW executing the code and determines response time by performing simulations

based on generated streams of instructions and evaluated with fixed input data. Nevertheless, this strategy requires, to build the tooling, a detailed knowledge of the underlying architecture. It also suffers from a lack of scalability, since fast hardware (HW) operations are simulated by a software (SW) that is orders of magnitude slower [Pim17].

SystemC-based simulation (SysC)

The SystemC C++ libraries [Acc] provide timing annotation mechanisms to modularly specify an algorithm and simulate its timing behavior with several levels of timing accuracy. The SystemC breakthrough is notably caused by a reduced simulation time with respect to Hardware Description Languages (HDLs) simulation, and a coupling with other system simulators, extending design interoperability [Men+17]. The good availability of general-purpose platform models on which to simulate the applications makes the usage of SystemC quite common for timing analyses [LIP]. However, the low level of abstraction of such models requires still detailed HW information [Moy14].

UML-based simulation (UML)

The Unified Modeling Language (UML) standard object model decomposes a computation into a set of objects. UML extensions such as SysML[Obja] and MARTE [Objb] are providing KPI specification semantics to feed an abstract simulation. In these DSE tools, latency estimation is performed by Register Transfer Level (RTL) or Transaction-Level Modeling (TLM) simulations [Gam+11]. In addition, UML-based simulations can exploit the benefits of the other application models, such as those based on dataflow (DF) [Gli+15]. However, MoA-based evaluations, which are focused on abstract KPI cost computation and can be used in the context of UML-based methods, can further increase KPI cost standardization offered by the plethora of UML-based DSEs works [Pel+18].

Dataflow model-based KPIs evaluations (DF)

DF MoCs have proven useful for modeling signal processing applications. When using a DF MoC, one of several alternative abstract models can be chosen to represent the computation of a functionality as a set of non-preemptive tasks which exchange messages with each other through First In, First Out data queues (FIFOs) [PL95]. Each DF MoC offers a different trade-off between application behavior predictability and runtime reconfiguration of the workload. Synchronous DF (SDF) [LM87] in particular makes it possible

to precisely define an execution iteration that is indefinitely repeated after an initial transitory phase. In that case, response time can be defined as *the time between the beginning of the execution of the firstly executed data source actor in the graph iteration and the end of the execution of the last data sink actor, where the actors are non-preemptive tasks and a data source actor (data sink actor) corresponds to a task that acquires (provides) determined input data (output data) of the DF network*. Latency evaluations can be computed from a DF MoC instance and a model of the HW architecture [Mat+19]. In that case, the accuracy of the timing analysis relies on the quality of these timing models, based on KPI estimates that define runtime behavior thresholds (such as Worst-Case Execution Time (WCET) or average execution time).

Model	**Pros**	**Cons**
Imp	cycle accuracy	detailed HW/SW information
SysC	high accuracy	limited HW model abstraction
UML	abstraction, flexibility	limited KPI cost standardization
DF	abstraction, modularity	lower accuracy

Table 2.1 – Comparison among application models in DSE context.

Table 2.1 summarizes advantages and disadvantages of the previously presented categories. Depending on the latency evaluation objectives and their requirements in terms of accuracy, the DSE methods based on the presented categories can lead to intractable evaluation problems and long times of development and analysis. This is mainly due to the amount of information (even not always available) required in this type of investigation, which implies scalability issues. Different design tools can be used depending on the constrained and optimized KPIs that have to be considered during development of the functionality. In particular, in order to achieve timing estimation and optimization for embedded systems, simulators working on different abstraction levels are exploited [ABA16; OB18; Pim17]. Among these tools, DSE tools are focused on high abstraction levels and designed for obtaining fast evaluation. Conversely, low-level analyses such as the ones performed by ISSs favour accuracy at the expense of an increased exploration time. For these reasons, having the possibility to include in the analysis a certain amount of information available depending on the degree of accuracy required (i.e. elasticity in the DSE) is desirable to efficiently exploit this trade-off and improve design productivity [AA18]. This is particularly true with large design spaces, where abstraction is needed

to reduce exploration complexity [Jon+18]. In this thesis, a method called PathFinder (PF), aimed at estimating latency in the DF context, is proposed (as described in Section 3.2). This choice is dictated by the fact that DF MoCs give advanced information on both computation and data transfers in the application, represented as a network of tasks that communicate with each other. This approach presents capabilities of modularity and abstraction that have been widely exploited in timing analysis (as explained in Section 2.3), including evaluation of certain undesired behaviours that may occur at runtime on an MPSoC [RDP17]. Moreover, it naturally fits for latency predictability in heterogeneous architectures [FW19]. For these reasons, the method for analysing system execution latency in this thesis is based on a DF representation of the application, and Section 2.1.2 and Section 2.1.3 propose an overview of the DF MoCs and the timing-focused tools based on these models respectively.

2.1.2 Dataflow Model of Computations

In literature, a plethora of DF MoCs has been defined. In particular, their semantics represent different modeling in terms of activation, execution, and interaction among tasks that describe the desired functionality. For this reason, each MoC implies specific features as reported in Table 2.2. Although other MoCs exist in literature (such as multidimensional models [ML02; Gam+08]), this table takes into account only a representative part of monodimensional MoCs, since target tools support this type of DF models (see Section 2.4). Kahn Process Networks (KPNs) [Kah74] consider a certain number of parallel tasks in communication with each other via unbounded FIFOs. Each task represents a specific amount of computation of the whole functionality, which, at each iteration, receives certain input data (called *tokens*) through the FIFO interface. In particular, FIFO channels writes and FIFO channel reads are non-blocking and blocking respectively. Other MoCs presented below derive from the KPN model, since this corresponds to the main family in the DF context [Pel+13]. As a specialization of the KPNs, Dataflow Process Networks (DPNs) [LP95] imply a sequence of executions (or *firings*) for each task (called *actor*). Such executions are completed without any interruption and enabled by a set of dynamic *firing rules* based on the presence of tokens into bounded FIFOs. SDF [LM87] specializes DPNs providing static firing rules, losing the dynamic datapath reconfiguration property. Indeed, consumption and production rates associated with the number of tokens are fixed and constant for all the executions. This leads to the advantage of improving design-time predictability leveraging on a simpler analysis of the model. As an extension of the SDF,

MoC	Reconfig.	Rate Variability	Determinism	Expressivity	Turing Completeness
KPN	N	N	N	high	Y
DPN	Y	Y	N	high	Y
SDF	N	N	Y	low	N
BDF	N	Y	N	high	Y
CSDF	N	Y	Y	low	N
HDF	N	Y	Y	low	N
SADF	Y	Y	N	medium	Y
PSDF	Y	Y	Y	medium	N
PiSDF	Y	Y	N	medium	N
SPDF	Y	Y	N	high	Y
BPDF	Y	Y	N	high	Y
EIDF	Y	Y	N	high	Y
CFDF	Y	Y	Y	high	Y

Table 2.2 – Comparison of the presented DF MoCs.

Boolean DF (BDF) [Buc93] involves further actors in order to control the token consumption and production rates. This control flow makes the BDF model Turing complete unlike its more specialized version SDF. Like BDF, Cyclo-Static DF (CSDF) [Bil+96] enables variable token consumption and production rates, representing another extension of the SDF. However, the control flow consists of a cyclic variation of the rates through a state modeled in the actors and initialized every determined number of firings. Other MoCs limited to applications that provide for the sequential execution of scenarios are the Heterochronous DF (HDF) [GBL99] and the Scenario-Aware DF (SADF) [The+06]. HDF supports changing the token consumption and production rates through Finite State Machine (FSM), whose state remains fixed during a firing, and during which the system can be considered as an SDF model. SADF extends SDF with the concept of scenarios, which imply the dynamic control of the rates. An alternative way to perform reconfiguration of the data rates at runtime in static MoCs is represented by the Parameterized SDF (PSDF) meta models [BB01]. Their strategy extends a DF model with dynamically reconfigurable hierarchical actors and considering parameters modifiable at runtime. With a similar approach, the Parameterized and Interfaced SDF (PiSDF), or πSDF, [Des+13] refines the control over the parametrization compared to the PSDF and merges the notion of interface [PBR09] that insulates the different levels of hierarchy in the application representation. This is managed by improving graph composition and adding dependen-

cies among parameters, whose relations are defined in a hierarchical tree. Other models that make use of parameters to set consumption and production rates are the Schedulable Parametric DF (SPDF) [FGP12] and the Boolean Parametric DF (BPDF) [Beb+13]. Nevertheless, these represent non-hierarchical reconfigurable generalizations of the SDF model. SPDF considers special actors in charge of modifying parameters after every specified number of firings, with the purpose of guaranteeing schedulability. BPDF leverages on a combination of integer and boolean parameters in order to dynamically control rates and communication channels respectively. An alternative to the parametric approach consists in other dynamic MoCs. With respect to the parametric-based MoCs, these models, such as Enable-Invoke DF (EIDF) [Pli+08] and Core Functional DF (CFDF) [PSB09], offer more efficient implementations with different degree of analyzability. These models are by nature less predictable than PiSDF. In the EIDF, consumption and production rates depend on the dynamic firing modes of the actors. These latter are handled by model specifications that check the activation requirements given the rates of current mode (*enable*), execute the requested functionality and determine a set of feasible modes for the next firing (*invoke*). CFDF corresponds to a subset of the EIDF, that provides for a restriction of the achievable next modes, from a series to a single and deterministic case.

In this thesis, the PiSDF MoC has been chosen for the availability of advanced tooling and the predictability of the application representations supporting reconfiguration.

2.1.3 Timing-focused Tools in the Dataflow Context

In literature, various tools depending on specific DF MoCs have been developed over the years. This section focuses on those tools that provide system-level analysis to optimize timing of functionality expressed with the DF models (see Table 2.3). MAPS [Leu+17] is a compilation framework that targets applications expressed with the KPN model. The main feature of MAPS consists of providing DSE in order to favour fast functional validation in systems with heterogeneous PEs, which are defined by their cost functions. Applications can be implemented in sequential C code or in its extension called C for Process Networks (CPN). Developed by Kalray, MPPA AccessCore [Kal; BMd12] is a commercial framework dedicated to MPPA multi-core systems. Applications are expressed as CSDFs through a C-based language called Sigma-C. PREESM [Pel+14] is a rapid prototyping tool that provides design simulation and verification for applications described as a PiSDF graph. The tasks of the desired functionality are specified as C functions and mapped on

DF Tool	MoC	Arch. Model	Input Specs	Availability
MAPS	KPN	cost funct. of target	C,CPN	accademic
MPPA AccessCore	CSDF	abstract	Sigma-C	commercial
PREESM	PiSDF	S-LAM	C	open source
Ptolemy II	KPN,DPN,SDF, BDF,CSDF,HDF, PSDF	abstract	various languages	open source
SDF3	SDF,CSDF,SADF	abstract	commands, C/C++	open source
SPIDER	PiSDF	S-LAM	C++	open source

Table 2.3 – Comparison of the main timing-focused tools based on DF MoCs.

a target modeled through the System-Level Architecture Model (S-LAM). In order to execute the application on the system, PREESM provides code generation for MPSoCs. Ptolemy II [Pto14] provides modeling and simulation of heterogeneous systems expressed as combinations of various DF MoCs. A functionality is specified through a hierarchy of application graphs, whose levels conform to a determined MoC, which is represented in some modeling language (such as Java or C). SDF3 [SGB06] is an open-source framework that offers analysis and simulation for SDF, CSDF and SADF MoCs. functionalities are described by using command-line tools and C/C++ Application Programming Interface (API). Nevertheless, like in Ptolemy II, code generation of the application prototype for MPSoCs is not provided. SPIDER [Heu+14] offers runtime management of applications specified as PiSDFs. The development of the functionality can take place by exploiting the PREESM environment, and the actors internal code is provided as C++ code.

In this thesis, PREESM and SPIDER have been chosen since they provide a design environment for scheduling and mapping parametric functionalities at design time and runtime respectively. In addition, both tools are open source and consider the S-LAM as an architectural model, which is suitable for describing heterogeneous MPSoCs with a high level of abstraction.

2.2 Design of Coarse-Grained Reconfigurable Architectures

Nowadays, a wide spectrum of processing systems capable of offering different trade-offs in terms of flexibility and specialization/efficiency can be considered for signal processing systems. Figure 2.1 illustrates the main HW platforms depending on these design properties.

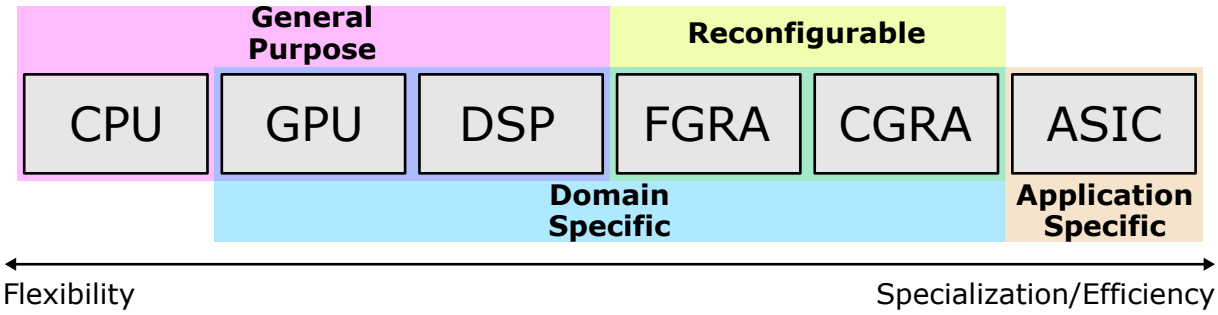


Figure 2.1 – Comparison among the main architectures for processing systems in terms of flexibility and specialization/efficiency.

Central Processing Units (CPUs) are general-purpose HW units capable of implementing arithmetic, logic, branching, and data transfer [HP17]. Since CPUs support any functionality, they provide the highest degree of flexibility. For this reason, this type of PEs is integrated in personal computers and most high performance computers. Nevertheless, CPUs do not represent an efficient solution when more strict requirements with respect to timing performance, chip area, energy/power consumption must be satisfied. In fact, each functionality is described through programming languages, then translated in a sequence of instructions (called *machine code*) suitable for the execution on these general-purpose units. The Instruction Set Architecture (ISA) of a CPU is extensive, with the objective to efficiently support a vast set of operations on integers, floats, vectors and matrices. In this context, the generic operations are performed in different pipeline stages that can imply specific amounts of clock cycles: instruction fetch, instruction decode, execution, memory and write back. This strategy leads to long runtimes, large circuitry sizes, and high energy/power demands. Indeed, the fundamental principle to which general-purpose systems comply is the adaptation of the functionality to the target architecture. Since the latter is generic and fixed, the efficiency is sacrificed in favor of the flexibility. Moreover, deep pipelines of instructions and multi-threading support make these architectures costly

in energy.

Although created for uses in specific domains, additional architectures can support general-purpose functionalities: Graphics Processing Units (GPUs) and Digital Signal Processors (DSPs). In particular, with respect to the CPUs, GPUs provide a more efficient handling for operations associated with image and video processing, for which they obtain the best performance. Indeed, their architectures are capable to exploit the typical data parallelism present in the applications of this domain. GPUs consider a pipeline capable of processing a large number of tasks and where the output of each task is linked to the input of the next task. Differently from CPUs, where the task is processed in stages in a pipeline divided over time and which exploits all processor resources, the GPUs provide for a distribution of HW resources between the stages of a pipeline divided into space [Owe+08]. Each part of the processor elaborates a stage and provides its output to the other part that takes care of the next stage. In order to make the best use of GPU, the code related to general-purpose operations has to be rewritten for a dedicated API or framework (such as the OpenCL library [Gro]). However, in addition to achieving the best performance on the GPU, the programmer must take care of coordinating the scheduling of the processing on the system processor and the GPU and the transfer of data between the memories associated with these units [HP17].

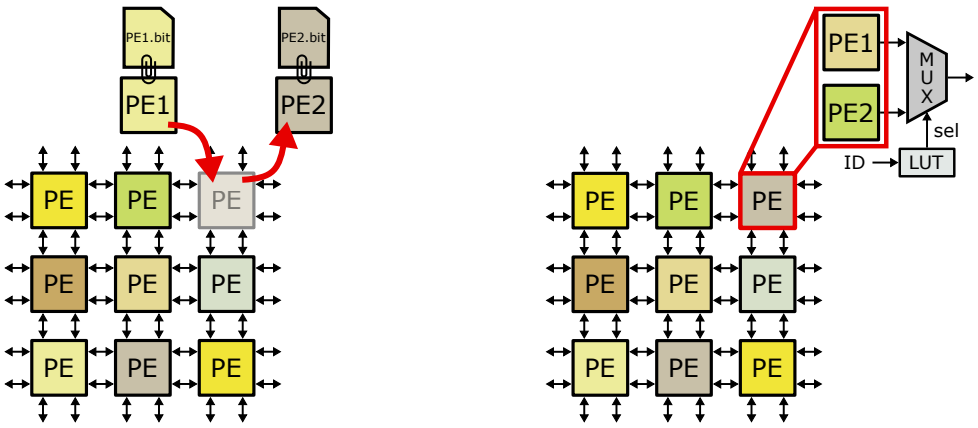
Since DSPs are also capable of performing generic functionalities, they can be counted among general-purpose devices. However, their ISAs are specialized for matrix, filtering and transformation operations (such as multiplication and accumulation) which are characteristic of signal processing. In this domain, DSPs can take advantage of the parallelism of data which can often be in both fixed and floating point formats. In order to obtain the best performance, as seen for the GPUs, the description of the applications is associated with specific compilers, and also requires the presence of signal processing operations and particular attention in the management of generic instructions.

In addition to the presented categories, there are others that respond to the need for high efficiency, despite a reduction in flexibility. The assumption on which they are based is reversed with respect to that one defined for general-purpose platforms. In this case, the architecture is adapted to the desired functionality. The devices that best denote this principle are the Application-Specific Integrated Circuits (ASICs). As depicted in Figure 2.1, they provide the maximum efficiency among the HW systems, since their implementation is dedicated to the execution of a single functionality. Only the HW blocks required to carry out this application are present in the circuitry. Generally, the architecture is

optimize to reduce the execution time related to the target algorithms, reason why in this case it is known as HW *acceleration* [Nur+16]. However, the optimization of these devices can follow different strategies based on the KPI selected in the development flow. This may determine different HW designs for the same functionality, which in any case remains the only executable in a system absolutely not flexible.

Reconfigurable devices offer a combination of the approaches that can be defined as *flexibility-oriented* and *efficiency-oriented*, typical of the general-purpose systems and HW implementations respectively. These consist of a set of PEs and connections that can be adapted at runtime [CH02], in order to perform a specific functionality or operating mode (called also *scenario*). The system adaptation is applied through a HW-level reconfiguration of logic blocks and interconnects [TB01]. These are deployed onto a logic substrate of a reconfigurable platform, the most common of which in the market is the Field-Programmable Gate Array (FPGA) [Res]. In literature, such platform is used to implement two main approaches based on Fine-Grained (FG) and Coarse-Grained (CG) reconfigurable architectures respectively [Che05]. These are defined depending on the degree of granularity associated to their PEs and connections. FG implementations are characterized by a bit-level reconfiguration of the logic substrate. This fine modification implies long adaptation times, although it favors high flexibility. On the contrary, since CG architectures involve word-level PEs, they leads to a reduction of the reconfiguration timing at the expense of flexibility. Figure 2.2 shows the fundamental difference between the FG and CG approaches. FG reconfiguration consists in totally or partially replacing the bit-level description of the architecture in the FPGA (see Figure 2.2A). On the other hand, reconfiguration in the CG approach is a selection of PEs or their configuration given the operating mode represented as an ID and chosen for the execution of the functionality (see Figure 2.2B).

In the last decades, systems composed of different platforms have been designed for optimizing flexibility and efficiency. Indeed, general-purpose architectures enable more flexible solutions with lower design effort and cost with respect to reconfigurable architectures or ASICs. Nevertheless, HW acceleration is sometimes necessary for computationally-intensive workloads in order to satisfy the use-case requirements [FW19], especially in terms of timing and energy/power consumption. Since this thesis is focused on the first aspect, and in particular on estimating latency for flexible systems, a heterogeneous architecture based on a combination of CPUs and CGR accelerators has been selected. In fact, this represents a trade-off between flexibility and efficiency in HW/SW co-design,



(A) FGRA (B) CGRA

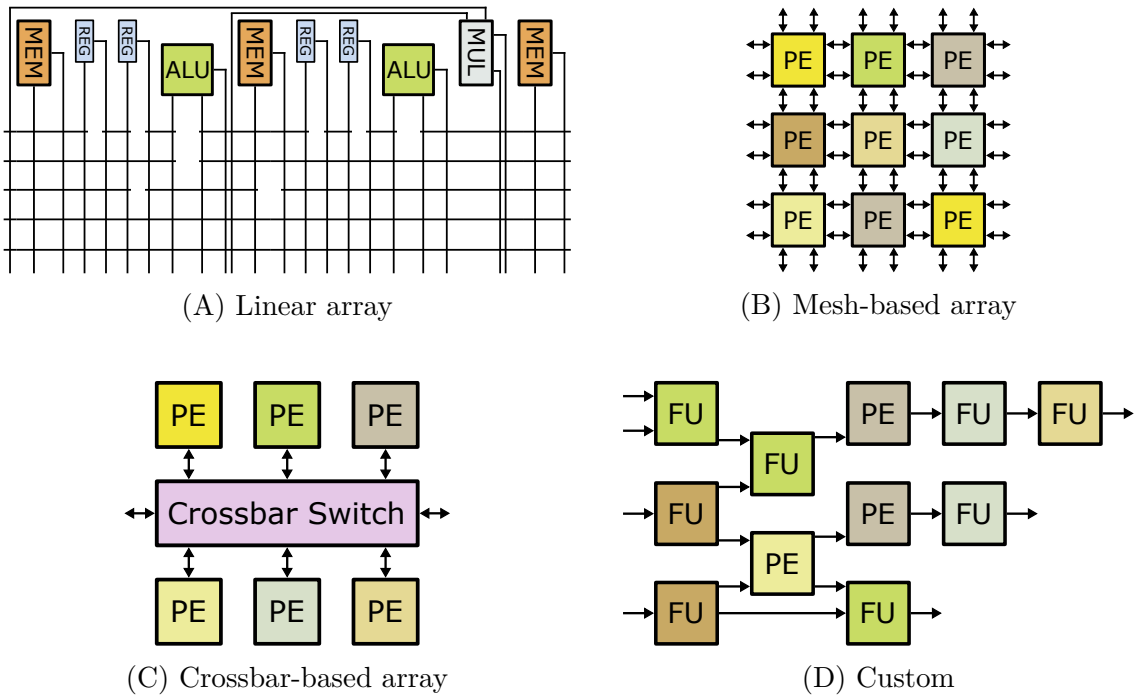
Figure 2.2 – Example of layout and reconfiguration in FG and CG reconfigurable architectures.

when avoiding high design overheads to gain performance.

2.2.1 Coarse-Grained Reconfigurable Architectures

Although CGR systems enable less flexibility with respect to the FG counterparts, they avoid the slow reconfiguration times required by the latter. In fact, they consists of bigger PEs that involve Arithmetic Logic Units (ALUs) and a relevant storage capacity. Since interconnects among PEs generally present the same level of granularity, Coarse-Grained Reconfigurable Architectures (CGRAs) make easier the place and route process than FGR Architectures (FGRAs) [Har01]. Moreover, CGR accelerators can be implemented on both ASIC or FPGA platforms, while FGRAs are specific to FPGAs. For these reasons, over the years various Coarse-Grained Reconfigurable Architectures have been proposed, which can be grouped into the main categories introduced in [Har01]. This classification focuses on the differences in terms of layout and connections of the PEs, distinguishing three different types: linear arrays, mesh-based arrays and crossbar-based arrays.

Linear arrays are often associated with the pipelined execution of the functionalities (see Figure 2.3A). Each PE is connected with its closest neighbors (predecessor and successor) in order to constitute linear arrays. In case of a combination of several PEs running in parallel, further routing resources are needed, which may involve switching blocks among multiple stages of the pipeline [Sma+14]. For these properties, CGRAs based on linear arrays can implement processor-like systems where instructions are processed in stages [HSM03].



(A) Linear array

(B) Mesh-based array

(C) Crossbar-based array

(D) Custom

Figure 2.3 – Different types of layout and connections used in CGRA design.

Mesh-based CGRAs are used in the most of the cases. As depicted in Figure 2.3B, these consider a matrix of PEs, each typically connected to its 4 or 8 neighbors. Implementations of this kind of systems have been proposed in [NKS12] and [PDM12]. In the former, the CGR architecture involves arithmetic and logical units for integer or fixed point numbers arranged as a 8-by-8 mesh and an additional line of storage elements. In the latter, the proposed system evaluates PEs composed of specific DSP and memory blocks of the vendor-specific FPGA on which this is deployed.

Less commonly used are CGRAs based on a crossbar switch implementing the connection among PEs (see Figure 2.3C). One of the first examples of this type of systems is organized with a communication structure consisting of a local and a global network, which respectively connect the 4 PEs (implemented as ALUs) grouped in a cluster and the clusters among each other [YR93]. A further implementation with different crossbar configurations has been proposed in [Ino+10].

In addition to the categories mentioned above and proposed in [Har01], CGR systems can integrate sets of heterogeneous PEs (e.g. DSPs, processors, etc.) and functional units (FUs) (e.g. adders, multipliers, etc.) and interconnects strictly necessary to the domain-specific functionalities (as shown in Figure 2.3D). In fact, since applications do not always

fit in CGRs with fixed elements, custom PEs capable of increasing the efficiency of the designed solution are often needed, and CGRAs are employed as co-processors for these general-purpose units.

2.2.2 State-of-the-art Tools for CGRA Design

Since features and granularity of CGRAs are highly varied, several works in literature address their design issues [CH02] [WWC16]. Indeed, complexity appears under different aspects such as: hardware (HW) design, platform dependency, resources mapping, optimisation, reconfiguration management. Although CGR acceleration can be achieved by writing HDL code manually, High-Level Synthesis (HLS) provides design automation through a high-level description of the desired functionality. The main approaches are based on a design toolchain with HLS or on a sole HLS tool. In both cases, the input specifications can be described by using the application models mentioned in Section 2.1.1.

Stefanov et al. [Ste+04] propose a KPN-based flow for a system composed of a CPU and an FPGA. The functionality is specified as a Matlab code and mapped upon the target architecture. In order to generate the implementation of the application starting from the Matlab specifications, the development of COMPAAN and LAURA tools has been required. Raffin et. al [Raf+10] present a framework named ROMA for the scheduling, binding and routing of reconfigurable accelerators for multimedia applications. The ROMA architecture is composed of custom reconfigurable PEs and the framework is specific to them, executing a fixed set of standard operations, thus with limited specialization. Voros et. al [Vor+13] leverage on a fixed CGR structure that comes along with its synthesis flow, and where 3 different types of PEs can be used for accelerating purposes. Such types of PEs corresponds to 3 heterogeneous reconfigurable architectures: i) a streaming-oriented CGRA based on an array of 16-bit PEs that exchange data through configurable channels, ii) a reconfigurable architecture for different application domains that is based on a combination of a RISC processor and a mid-grain reconfigurable datapath, iii) an embedded FPGA suitable for FG Reconfigurable (FGR) algorithm or arbitrary logic implementation. Among tools based on the MPEG-RVC standards and the Cal Actor Language (CAL), Open RVC-CAL Compiler (ORCC) [Sir+10] is the most commonly used compilation infrastructure. It provides generation of descriptions for software (SW), hardware (HW) or mixed designs leveraging on an intermediate representation of the DPN MoC. Indeed, this latter can be used in DSE and vendor-specific FPGA-based HLS tools (such as Turnus [CMJ13] and Xronos [BMJ13] tools). Beaumin et al. [Bea+10] adopt

designs based on MPEG-RVC-cal for building a custom CGRA, where actors are directly mapped onto different PEs and communicate through FIFOs. The reconfiguration consists of modifying DF actor currently executed by a PE. However, design automation is not supported. Multi-Dataflow Composer (MDC) is a tool for the creation and the DSE of multi-functional reconfigurable architectures [PSR15]. This tool has been previously integrated with ORCC in order to optimize DF specifications for the generation of CGR HDL. Then, it has been combined with the Xronos HLS to build a vendor-specific DF-to-HW design environment [Sau+14]. Ciobanu et al. [Cio+18] propose an open design platform named EXTRA and based on a web-oriented user interface for the development of accelerated applications for High Performance Computing (HPC) systems. Depending on the DF specifications and the HW requirements, EXTRA generates reconfigurable architectures implemented in FPGA devices. Synflow Studio [Syn] is an Integrated Development Environment (IDE) for the HW design of functionality described as a task networks. This is specified by a C-based high-level code, written in a proprietary DF language named Cx. The code compilation translates into a HDL (Verilog or VHDL) which can be synthesized and simulated by several design tools for multi-vendor FPGA devices. CAPH [SBA13] represents another HLS framework based on DF MoC and specialized for streaming applications. Applications are described as a DF network through functional expressions of the CAPH language. The generated HDL can be adopted for synthesis in ASIC and FPGA platforms. The Xilinx Vivado HLS [Xil] provides generation of HW Intellectual Properties (IPs) targeted for Xilinx FPGAs from a C-like description of the functionality. Moreover, the design environment provides functional verification and DSE after the synthesis phase. The Intel FPGA SDK for OpenCL [Int] is a development suite for systems with Intel CPUs and FPGAs. Applications are specified in the OpenCL language, a variant of the C language widely used in GPU programming. Starting from C/C++/SystemC specifications, Catapult HLS [Gra] provides RTL designs for ASIC and FPGA platforms. However, in order to perform the RTL synthesis, vendor-specific tools are required.

Table 2.4 reports a qualitative comparison among the HLS-based design solutions. This table highlights the characteristics of the HLS toolchains respectively in terms of: application model as introduced in Section 2.1.1, ability to re-map the functionality on different HW components at runtime, the possibility of generating the HDL describing the CGRA, the platform on which the generated accelerator, the heterogeneity and the connection topology of the PEs belonging to the system, and the availability to use the toolchain. In order to favor IP-level latency predictability in platform-agnostic design of

CGR accelerators, a new trade-off named DataFlow-Functional (DFF) HLS [Rub+19] is proposed in this thesis. It builds on the properties of two open-source DF-based tools mentioned above: MDC and CAPH (for more details see Sections 2.4.2 and 2.4.1 respectively). In Section 3.1, HLS design approaches and motivations at the base of the novel methodology are discussed. Section 4.1 presents the design flow of the DFF HLS and an assessment in the context of video coding with respect to the main commercial tools (Xilinx Vivado HLS and Intel FPGA SDK for OpenCL).

HLS Toolchain	**Appl. Model**	**(Re-) Map.**	**HDL Gen.**	**Target Platform**	**PE Heter.**	**Topology**	**Toolchain Availability**
[Ste+04]	DF	Y	Y	FPGA	Y	custom	accademic
[Raf+10]	Imp	Y	N	custom device	N	fixed	accademic
[Vor+13]	Imp	Y	Y	custom device	Y	fixed	accademic
[Bea+10]	DF	Y	Y	FPGA	Y	fixed	open source
[Sau+14]	DF	Y	Y	Xilinx FPGA	Y	custom	open source
[Cio+18]	DF	Y	Y	FPGA	Y	custom	open source
[Syn]	DF	Y	Y	FPGA	Y	custom	open source
[SBA13]	DF	N	Y	ASIC, FPGA	Y	custom	open source
[Xil]	Imp	Y	Y	Xilinx FPGA	Y	custom	commercial
[Int]	Imp	Y	Y	Intel FPGA	Y	custom	commercial
[Gra]	Imp	Y	Y	ASIC, FPGA	Y	custom	commercial
[Rub+19]	DF	Y	Y	ASIC, FPGA	Y	custom	open source

Table 2.4 – Overview of the main state-of-the-art toolchains that provide CGR acceleration through HLS.

2.3 Timing Analysis on MPSoCs

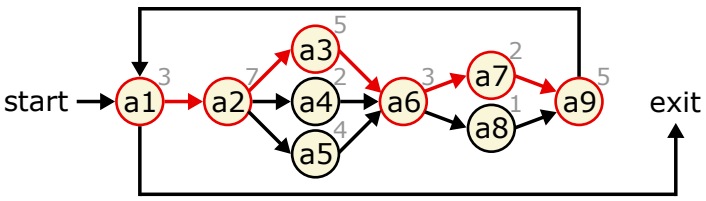
Knowing timing behavior of a desired functionality is an important premise in the effectiveness and efficiency of system design, especially in the presence of time constraints. Basing on the requirements for running a functionality, development can take several paths depending on the choice of the application model and the type of architecture.

However, the problems associated to the timing estimation and the strategies employed in solving them belong to a common context. In particular, studies concerning real-time systems are very important, since they address very strong requirements. In fact, works of this research area are useful in understanding the problems related to the estimation of execution times. Time estimation is specifically difficult when an application is mapped to an architecture with multiple PEs. For this reason, this section proposes an overview of time estimation research areas. At first, the main concepts at the base of timing analysis are discussed in Section 2.3.1. Indeed, these refer to the techniques used in the literature for the evaluation of the response time. The state-of-the-art strategies that make use of one or more of these techniques are described in Section 2.3.2.

2.3.1 Fundamental Concepts in Timing Analysis

The main strategies adopted in timing analyses aim at evaluating the WCET and handling the limitations due to task concurrency in a finite architecture. These strategies are based on the use of techniques such as: WCET analysis, schedulability analysis, resource arbitration, and cache partitioning.

WCET Analysis



(A) CFG with timing and highlighted WCET path

Input Data and Iterations

$t_{path} = 25$

$t_{header} = 3$

$iterations = 100$

WCET calculation

$WCET = t_{header} +$

$+ t_{path} * (iterations - 1) =$

$= 3 + 25 * 99 = 2478$

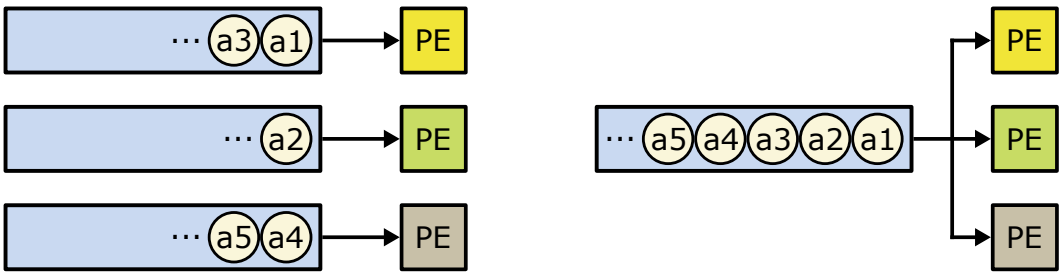
(B) WCET calculation

Figure 2.4 – Example of CFG in WCET analysis.

WCET defines a deadline within which the execution of a single-task application in a single-core platform is guaranteed [Wil+08]. However, calculating this bound is challenging, since it depends on the characteristics of the functionality and architecture [MTT18]. In fact, the input data values impact the operations of the tasks, which can be associated with different control flow branches and certain delays due to the memory hierarchy of the architecture. Therefore, WCET identification is a complex process that has been

widely explored in literature. The common approaches can be classified into static and dynamic evaluations [Lv+16]. In order to estimate WCET, the static evaluation methods are based on representing the executable program of the task as a Control-Flow Graph (CFG). As depicted in Figure 2.4A, a CFG is composed of instructions (vertices) and control flow statements (links). Analysing all paths present in the CFG (the set of paths must be finite), ranges for input data and iterations are extracted or provided by the user. Selected paths and ranges determine the degree of flexibility of the admitted behaviors of the application. This information is therefore combined with an abstract model of the architecture or a model checker, in order to compute the bounds of the instructions. Since WCET is computed as a sum of these contributions (see Figure 2.4B), the accuracy of the used model heavily affect the analysis in terms of estimation quality and complexity. On the other hand, achieving WCET through dynamic, or measured-based, methods consists in measuring end-to-end execution latency of a subset of task scenarios and selecting the maximum measure as WCET [Wen+08]. The measurements are performed by using either the target hardware (HW) or an ISS. In the case of measuring execution times without the use of an ISS, this fact avoids the detailed knowledge about the ISA specifications and system features. However, an underestimated WCET is generally provided by dynamic analyses because of a limited case coverage. For this reason, more refined hierarchical methods that compute WCET by analysing measures of parts of the task (mainly, CFG blocks) combined with all possible paths (extracted from the CFG as in the static approach) have been proposed. One may note that any approximation on either application or architecture jeopardizes the WCET estimate reliability.

Schedulability Analysis



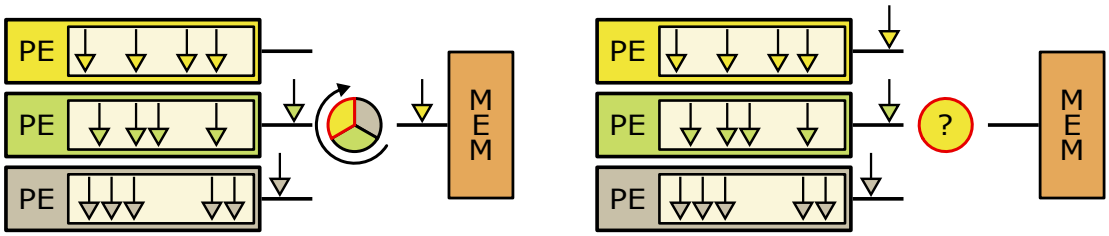
(A) Partitioned scheduling (B) Global scheduling

Figure 2.5 – Example of partitioned and global scheduling.

In order to determine the response time of a complete application composed of several

tasks, a schedulability analysis is required. Typically, this analysis takes as input WCET (and/or Best-Case Execution Time (BCET)) bounds of the tasks and give as a result the Worst-Case Response Time (WCRT) [Axe+14]. Scheduling algorithms have been proposed in the analysis depending on the policy chosen for task resource sharing. These can be grouped in two main classes *partitioned* and *global scheduling* [Gua+11]. The former evaluates a fixed mapping of each task upon a single PE, not allowing any task migration (see Figure 2.5A). In the latter, tasks can be performed on different PEs, either during its execution or between its distinct executions (see Figure 2.5B). Both classes are often specified depending on a further property of scheduling algorithms: the possibility to interrupt task execution and resume the task later (*preemption*). *Preemptive* and *non-preemptive scheduling* can be applied depending on the selected design trade-off. In addition, information associated with the task priority, operating-system overheads, and shared-resource delays can be integrated into the schedulability analysis. In this way, timing requirements of tasks and application in a specific context can be verified [AP14].

Resource Arbitration



(A) TDMA (B) RR

Figure 2.6 – Example of memory arbitration using static and dynamic policies (respectively by considering TDMA and RR).

In a CPS context where the system has many sensors and actuators, applications involve multiple tasks that concurrently run on MPSoCs. Since WCET-based estimates consider tasks executed separately, delays due to accessing shared resources (such as buses and memories) lead to strongly incorrect analyses. For this reason, evaluations aimed at avoiding this issue are required. A typical strategy is to remove the effects of interference by adopting predictable shared resource arbitration mechanisms [Mai+19]. Such arbiter handles conflicts in task requests by guaranteeing them access to the bus and memory one at a time. However, *implicit* accesses (e.g. related to the cache-coherence protocol) increase uncertainty in static task management [Kel+14; HKP17]. The main

techniques used in this context are respectively called: *time-driven*, *event-driven*, and *hybrid* [Abe+13]. Time-driven arbitration consists in a predefined scheduling of the access requests and in the exclusive assignment of the shared resource for a determined time slot. This policy, commonly known as Time Division Multiple Access (TDMA), distinguishes the execution times due to computation and those deriving from concurrent access to shared resources. Figure 2.6A shows an example of memory arbitration with a TDMA policy, where in particular each PE periodically accesses the resource in a time slot of the same duration. Event-driven arbitration considers runtime decisions based on the access history for determining access grants. The main policies respectively evaluate, as a worst-case scenario, the circumstance where the maximum number of access requests are allowed before (Round-Robin (RR)) and simultaneously (First-Come First-Served (FCFS)) with respect to a specific request. Figure 2.6B illustrates the counterpart to Figure 2.6A for the RR policy, where slot assignment and length are decided dynamically. Hybrid arbitration combines the time- and event-driven arbitration depending on static or dynamic arbitration periods respectively.

Cache Partitioning

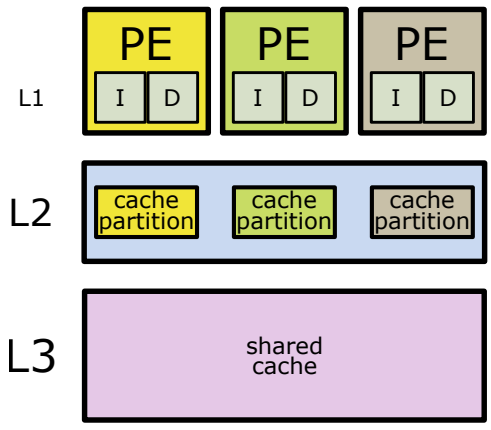


Figure 2.7 – Example of 3-levels cache hierarchy with partitioning.

In order to reduce data access time, various levels of cache are used in an MPSoC memory hierarchy. In general-purpose architectures these levels are 3 or 4, and are referred to as L1, L2, L3 and L4 on the basis of an increasing order by size. Each level of cache can be assigned exclusively by a single PE (*private cache*) or can be shared between different PEs (*shared cache*) (see Figure 2.7). Private caches provides data access with a lower latency with respect to the shared caches [DS07]. On the other hand, large shared

caches offer higher efficiency for the same chip area than small private cache sets. For these reasons, only the L1 level cache is often assigned privately, sharing the other levels of the hierarchy [Gra+15]. Nevertheless, parallel accesses to the cache resources create delays due to the inability to handle requests in a completely parallel way, leading to a pipelining of the accesses. Moreover, since cache levels are limited in size, not all required data may be available. This fact implies the use of a data replacement policy aimed at keeping only the most important data for a given time window. However, although access to data is favored through data replacement policy algorithms and pipelining of accesses can regulate parallel requests, accurate and efficient timing analyses remain challenging [Abe+13]. For this reason, a recent trend is to associate a specific part of the cache to each owner PE. In this manner, each memory block is replaced according to the need of the same PE. In this context, the partitioning methods can build their decisions based on HW or SW implementations using exclusive access to a determined number of ways, sets, or blocks of the cache [Mit17]. Especially in application for real-time systems, this choice is supported by a task characterization (provided by a WCET analysis), in order to count the potential cache hits [Abe+13]. Other approaches, based on previous task behavior, rely on dynamic partitioning to improve performance. These methods take into account the reconfiguration time needed to deeply modify system behavior. A further strategy to gain performance considers the optimization of the scheduling. In fact, the information about cache accesses can be exploited to modify task priority in static and dynamic scheduling. One may note that all practical scheduling problems are NP-complete and thus require heuristics and non-optimal decisions.

2.3.2 Main Strategies in Timing Verification

Focusing on the timing analysis of systems with multiple PEs, the survey presented in [Mai+19] classifies the state-of-the-art into four general research categories, as follows.

Full Integration (FI)

Works based on Full Integration leverage on the complete information regarding the execution of the tasks. The analysis is based on WCET, using model checking and abstract processor states. In particular, potential interference due to the simultaneous requests of the shared resources is taken into account in the evaluation. In addition, no particular relevance is dedicated to the exploration of various strategies regarding task scheduling,

which is considered known, non-preemptive and fixed with synchronous task releases with small or no jitter.

In literature, efforts have been focused on the evaluation of interferences based on the type of shared resources considered in the analysis: memory, bus and a combination of both. Works focused on the memory interference examine the cache state through models deriving from abstract assumptions or model checking [Gus+10; Lv+10; YPV15; NS16]. Works aimed at evaluating bus interferences integrate an accurate contribution model of the related interference in the WCET analysis, assuming shared memory interference as negligible or handled by cache partitioning or resource arbitration itself [Rih+15; JHH15; JHH16]. Other works deal with both types of shared resources [DN12; Bon+12; KM15].

Although FI promises precise estimations, generally it suffers from a high complexity and a low scalability, because HW real architecture must be known and precisely modeled analysis required in its methodologies. Moreover, the provided estimate relates to an exact context, since the integrated interference depends on how concurrency affects the access to the shared resources in given time windows.

Temporal Isolation (TI)

Temporal Isolation separates task processing over time in order to avoid contemporaneous usage of the shared resources, mainly the memory bus, which is considered as the main source of interference. In order to achieve this, different methods can be applied.

Solutions based on a *phased execution model* refactor the code of the task, identifying two or three main phases [Yao+12; Bak+12; WP14]. As a first step, data and instructions loading from main memory to private cache is performed. The second phase consists of executing task computation by taking advantage of the data and instructions availability. As a third phase, the data storing time in the main memory can optionally be evaluated. Exploiting this model, a proper task scheduling can avoid overlapping memory access phases.

Another strategy aims at handling accesses to the memory bus by using *memory bandwidth regulators* [Now+14; Man+17; FUU18]. These periodically assign a limited bandwidth to each PE. In this way, tasks running onto different PEs can access the memory with independent flows. However, in the case that a PE requires more bandwidth than the assigned one, this is forced to wait for the next period, even if other PEs are idle.

In order to provide temporal isolation, a further method evaluates *offline scheduling*

decisions [Gia+15; Per+16; CC18]. The set of scheduling decisions is statically generated to be used as is at runtime to manage access to shared resources. Nevertheless, specific operating modes or dynamic code modifications may require a regeneration of these specifications.

Other works make use of *hardware isolation* mechanisms that remove or limit the dependency of the interference on the co-running tasks. Common techniques involve resource arbitration (by using TDMA or RR) or cache partitioning [Ros+07; YKS11; OLF17]. In addition to requesting specific hardware designs, the implementation of Temporal Isolation often implies issues in effective bandwidth handling or pessimistic worst-case analyses.

Such approaches provide context-independent estimations, since the interferences due to the co-running tasks is bounded or removed in the analysis. In addition to improving predictability of the access requests, TI leads to a simpler timing evaluation than FI analyses. However, the lower HW utilization penalizes the performance in running the whole application, especially when a full isolation is applied [Mai+19].

Integration into Schedulability Analysis (IS)

Integration into Schedulability Analysis accounts additional delays due to the presence of co-running tasks and to the policy of accessing shared resources in the schedulability analysis. This integration is considered with respect to a model based on the WCET of each task in isolation. As a result of this combination, WCRT evaluation can be obtained. In literature, works that deal with sharing of single or multiple resources are present. These can be grouped depending on which they focus: interconnects ([Pel+10; Sch+11; SE11]), memory ([Kim+16; XAP17]), or multiple resources ([Dav+17; And+18]). For the mentioned reasons, the IS approach is characterized by highly flexible analyses, since it is compatible with various assumptions regarding different types of scheduling, arbitration, and shared resources. In particular, with this approach, analyses either context-dependent or context-independent are feasible. Indeed, the analysis can evaluate the behavior of actual co-runners or be based on worst-case considerations that include any co-runner. In addition, the main advantage of IS methods derives from modeling interference with a time slot for the resource demand that is equal to the task response time. This strategy reduces the pessimistic evaluation of the WCRT compared to the sum of worst-case delays associated to shorter time windows. Moreover, more precise analyses are provided when information about potential concurrent tasks and the amount of resource access requests in the task time slot are included. Nevertheless, an issue of circular dependency requires to

be handled. Indeed, execution times of the tasks depend on resource contentions, which in turn are based on the time slot of the tasks under evaluation. This can be solved for practical-sized systems with a quasi-polynomial complexity degree. However, works based on the IS approach effectively manage analysis with serialized access to the shared resources, at the cost of excluding their overlapped usage.

Mapping and Scheduling (MS)

Mapping and Scheduling combine a timing analysis with mapping and scheduling solutions to guarantee timing constraints. Mapping consists in assigning tasks to PEs and data to specific memory sections. On the other hand, the order and management of tasks are addressed by the scheduling. These two processed can lead to various solutions in terms of efficiency and their decisions are based on analyses with a different degree of computational complexity. For this reason, methods aimed at obtaining of optimal or sub-optimal mapping and scheduling are present in literature. These can be classified depending on the optimization strategy as follows.

A common method consists of converting the mapping and scheduling processes into a Integer Linear Programming (ILP) problem [Bec+16; Chi+16; ZWN17]. ILP provides an optimal solution for mapping and scheduling, based on the trade-off chosen for a function that depends on variables related to the problem (such as number of PEs, communication delay and order of tasks). In particular, all variables are limited to the domain of integers, and the function to optimize as well as the non-integer constraints are linear. Instead, if some decision variables are not discrete, the problem is known as Mixed-Integer Linear Programming (MILP).

Other works address a further problem when optimizing task partitioning, called the Bin-Packing (BP) problem [NP15; Xu+16]. BP consists in packing n objects (tasks) of different sizes in the minimum number of bins (PEs) of capacity C. Since this problem is strongly NP-complete, heuristics (such as next-fit, best-fit, worst-fit) have been developed to achieve close-to-optimal solutions quickly.

Additional strategies are respectively based on programming paradigms or on algorithms [Che+16; SS17; SI18]. In constraint programming, the solution of the mapping and scheduling problem is deduced from a relationship between variables. Dynamic programming can be used to reduce the complexity of the problem by dividing it into simpler sub-problems recursively. Finally, genetic algorithms, inspired by the natural selection process, are often used to solve such non-convex optimization problems by generating

random solutions and keeping the best ones.

The MS strategy aims to solve a wide issue: the fulfillment of the timing requirements and the searching of a valid mapping and scheduling solution at the same time. In FI, TI and IS approaches, mapping and planning are in fact generally taken as pre-established inputs to be provided for the analysis of the timing. However, mapping and scheduling are highly complex problems, and it is difficult to clearly evaluate the impact of the decisions on which a MS solution is based in terms of reliability and accuracy. In addition, MS works suffer from scalability issues, as well as strong dependence on application and architecture characteristics (such as the number of tasks and PEs respectively). For these reasons, they tend to not provide a complete solution for the entire issue faced by their approach.

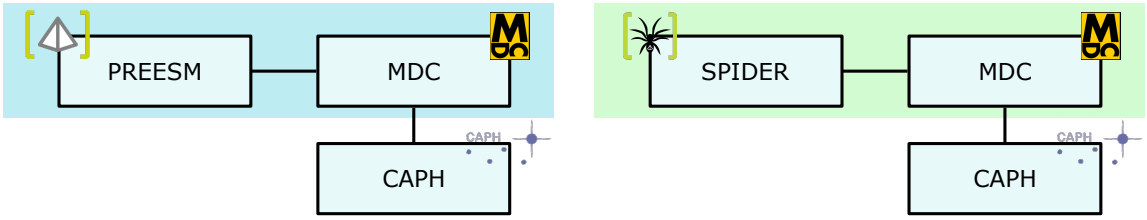
Table 2.5 compares the PathFinder objectives (presented in Section 3.2.4) to the properties of the previously presented timing verification categories. Assets are shown in green, while the light and heavy drawbacks are in orange and red respectively. Although the works of the reported publication categories provide early performance evaluations, they are based on complex analyses that are limited by the nature (mainly the size) of the considered applications and architectures. On the other hand, in order to benefit from the provided analyses, unambiguous and precise specifications for the application and the architecture are required. Moreover, existing timing verification methods require advanced knowledge of the software (SW) and hardware (HW) scheduling and features that are not always available in practice. PathFinder provides a novel trade-off among the listed properties. Indeed, the proposed method reduces the HW/SW information required by the state-of the-art strategies by estimating task execution times from platform measurements and Longest-Latency Path (LLP) analysis, enabling response time evaluation of DF-based applications mapped in a free way upon a wide range of target architectures. In addition, the proposed method aims at supporting strategies based on Model of Architectures [Pel+18], since PathFinder provides the response time activity of the functionality that can be used to build an MoA. With respect to the PathFinder, the main advantage in evaluating timing with MoAs is in estimating response time at design time on a range of architectures, at the expense of a loss of accuracy (as shown in Section 6).

Property	FI	TI	IS	MS	PF	MoA
Complexity	high	medium	medium	high	medium	low-to-medium
Scalability	low	medium	medium	low	medium	high
Accuracy	high	high	high	medium	high	low-to-medium
HW usage	medium	low	low	medium	high	high
Expressivity	medium	low	high	medium	high	high
required SW details	high	high	high	high	low	low
required HW info	high	high	high	high	low	low

Table 2.5 – Design properties of the timing verification approaches.

2.4 Tools used in the Proposed Design Flows

In this section, a description of the mentioned tools considered in the design flows is presented. These tools are chosen within the H2020 CERBERO consortium for their complementarity for building a model-based design method of MPSoCs with Coarse-Grained reconfiguration for signal processing applications. Figure 2.8 shows the connections among tools and their use for design time and runtime support.



(A) Design time support (B) Runtime support

Figure 2.8 – Overview of the tools utilized in this thesis. Tools chosen for design time and runtime support in the context of the H2020 CERBERO project are represented into the colored boxes.

2.4.1 CAPH

The Caph just Ain't Plain HDL (CAPH) [1] [SBA13] is an open-source, domain-specific framework for the specification, simulation and implementation of signal processing applications based on a dynamic DF MoC. Figure 2.9 depicts the structure of the CAPH toolchain. Applications are specified as dataflow networks (DFNs) using a higher-order, polymorphic functional language based on Objective Caml. The behavior of each DF

1. `https://github.com/jserot/caph`

actor is defined as a set of transition rules using pattern matching on structured data, resulting in improved abstraction capabilities. The CAPH toolchain provides graphical visualization of DFNs; code simulation with trace facilities; and HLS producing SystemC code for simulation and resource-monitoring purposes, as well as platform-agnostic ready-for-synthesis VHDL code.

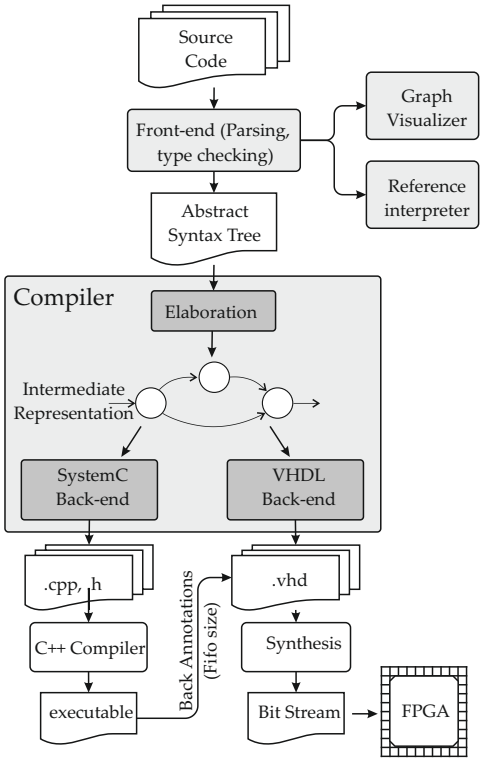


Figure 2.9 – CAPH toolset overview.

2.4.2 MDC

The Multi-Dataflow Composer (MDC) [2] is a toolset for the design and development of custom CGRA based on a set of DFNs: the DPN MoC [Pal+17]. As shown in Figure 2.10, the MDC design suite is composed of two main sub-components:

— the *Merging Process (MP)*: a dataflow (DF)-based model-to-model compiler that, given an input set of DPNs describing the functionalities to be executed in hardware (HW), generates a high-level multi-dataflow (DF) DPN [Sau+15a] of the system leveraging on datapath merging techniques [Sou+05];

2. `https://github.com/mdc-suite`

— the *Platform Composer (PC)*: a dataflow (DF)-to-hardware (HW) synthesizer that, given the mentioned multi-dataflow (DF) specification, the HW description of the DF actors and the protocol used for communication among them, generates a CGRA.

MDC offers also other features that optimize the generated systems and favor their integration in real environments, such as:

— a structural profiler [PSR15] that, taking into account the low level feedback coming from a priori synthesis of the generated platform, is capable of identifying the optimal multi-dataflow (DF) configuration depending on a set of metrics (area, power, frequency);

— a power manager [Fan+16; Fan+15] that automatically sets power saving strategies, such as clock- and power-gating at system modelling level;

— a rapid prototyper [Sau+15b] embedding the generated CGR systems onto ready-to-use platform-dependent IPs (for Xilinx devices).

Before the integration work proposed in Section 4.1 of this thesis, MDC was specific to the RVC-CAL language and the actor communication protocol was hardwired on the resulting CGR architecture. Now users define custom communication protocols, and MDC accordingly implements the handshake among PEs.

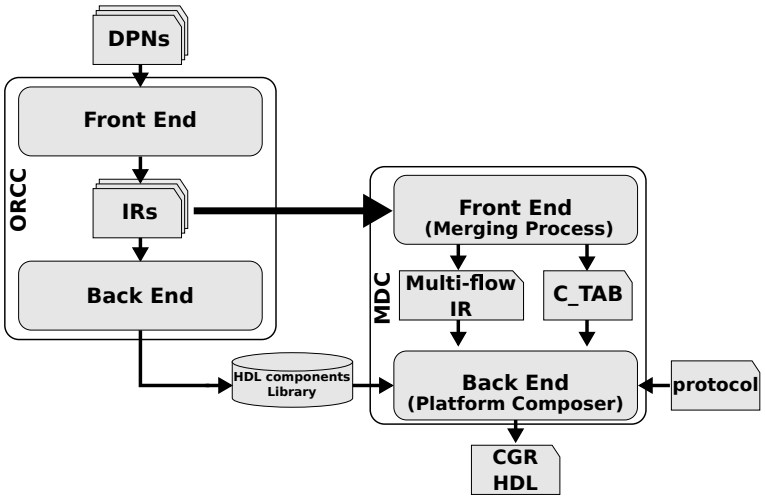


Figure 2.10 – Overview of the MDC design flow.

2.4.3 PREESM

The Parallel and Real-time Embedded Executives Scheduling Method (PREESM) [3] [Pel+14] is an open-source rapid prototyping tool for signal processing applications described as a PiSDF graph and mapped upon heterogeneous multi/many-core embedded systems. Its main functionalities currently are:

— A fully automated mapping of actors to multiple processing PEs with the objective of optimizing statically the execution latency and load balancing;

— A state-of-the-art and fully automated optimization of the generated application memory footprint.

Figure 2.11 illustrates its design flow. This starts from the creation of the PiSDF model, that specifies the desired functionality, and the S-LAM graph, that represents the architectural model. In addition, a scenario associated to such inputs summarizes the characteristics of the system to be implemented. Depending on the scenario information, PREESM applies a single-rate (sr) transformation to the PiSDF in order to simplify scheduling and mapping potentially at the cost of large graph management. At this point, static scheduling (List or Fast [KA99]) and memory management are performed. Output of these phases are the C code generation of the system and the Gantt visualization of the achieved solution respectively. The execution is self-timed, with one thread on each PE, synchronized by inter-PE communication. Finally, by adding the actors specification, the C code compilation and the system execution can be evaluated.

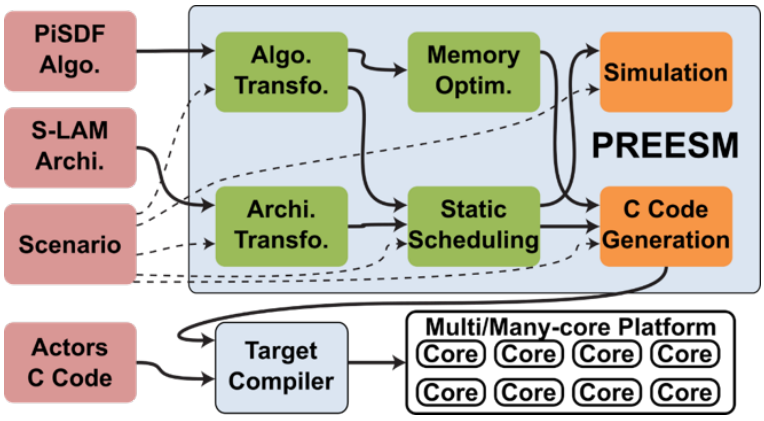


Figure 2.11 – Overview of the PREESM flow.

3. https://preesm.github.io

2.4.4 SPIDER

The Synchronous Parameterized and Interfaced Dataflow Embedded Runtime (SPIDER) [4] is a runtime manager for applications described through the PiSDF MoC and executed on heterogeneous multi-core architectures [Heu+14]. When compared to DPN, PiSDF increases processing and communication predictability, serving as input information for multicore and multisystem partitioning, at the cost of some expressiveness, i.e. some non-deterministic application behaviors cannot be modeled with PiSDF. The SPIDER runtime is currently available for ARM/Linux-based architectures, Intel x86 platforms, Keystone II architectures from Texas Instruments, and MPPA256 many core from Kalray. In order to ensure independence between application and platform levels, the SPIDER runtime structure consists of the following layers:

— the *Application Layer*: that is composed of dataflow actors and PiSDF specifications describing a stream processing application;

— the *Runtime Layer*: the core of the runtime manager consisting of a master part called Global Runtime (GRT), that handles scheduling and mapping of the application, and slave elements named Local Runtimes (LRTs), that execute the processing of the actors depending on the current scheduling decided by the GRT;

— the *HW Specific Layer*: this layer is a platform-dependent component designed to manage the inter-core communication and synchronization.

Figure 2.12 shows the execution scheme of SPIDER. The GRT (master) schedules the application (1) and sends the execution order based on the mapping decisions (2). The LRTs (slaves) deal with the execution of the actors present in the dedicated job queue (3). Jobs are data structures containing the information about synchronization, data and code of the actors to properly perform one instance of an actor in a specific slave. The LRT can be implemented over general- or special-purpose processors, as well as accelerators. During code execution, LRTs exchange data tokens through a pool of data FIFOs (4). Once the processing of the actor is completed, the LRTs send new parameter values (if any) to the GRT (5). Indeed, a parameter value can be set dynamically by a configuration actor mapped in a LRT, influencing the algorithm execution. Moreover, the GRT receives also the execution traces (the actor start and end times based on the same timing reference) by the LRTs (6).

4. `https://github.com/preesm/spider`

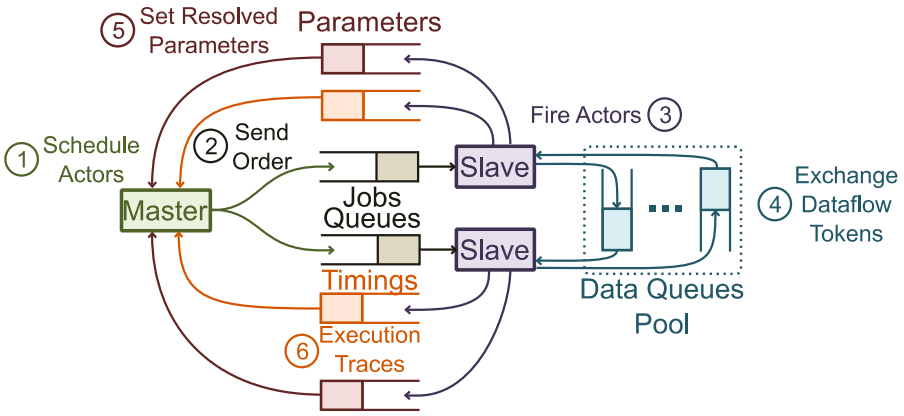


Figure 2.12 – The SPIDER runtime internal scheme.

2.5 Chapter Remarks

In this chapter, the state-of-the-art strategies used in the design of a parametric functionality have been presented. A focus has been put on design based on signal processing application models and CGR hardware. Indeed, these represent a valid trade-off in order to obtain a system with flexibility and efficiency. In particular, the description of the functionality in a SW-oriented environment such as PiSDF and in HW-oriented environment such as CGRA provides the operating mode reconfiguration through the use of parameters. For these reasons, the tools described in Section 2.4 have been integrated in the design flows related to the contributions of this thesis. For taking a design decision, a system designer or an automated DSE procedure require an early knowledge of the impact on performance of the different alternatives. State-of-the-art timing analysis methods are either over simplifying MPSoC architecture or requiring excessive knowledge of the underlying hardware. PathFinder aims at proposing a lightweight alternative to this state of the art for CPS design.

In addition, common concepts and approaches in the evaluation of the response time of applications mapped on architecture with multiple PEs have been reported. Specifically, such approaches represent the pillars of the timing verification for the HW/SW contexts, although they have been mainly assessed for SW-based applications running on general-purpose systems with multi or many homogeneous PEs. In the specification of the functionality, they make extensive use of DF MoCs, which favour predictability and abstraction in timing analyses. Based on the same strategy, our proposed PathFinder and MoA-based approach have been briefly introduced and compared to the state of the art as

alternative methodologies. These correspond to another trade-off for early system timing analysis aimed at reducing the amount of HW/SW information required in the traditional approaches.

Chapter 3

Latency Estimation in Hybrid Flexible HW/SW CPSs: Problem Definition and Objectives

In this chapter, our objectives on latency estimation in hardware and software designs aimed to flexible solutions are presented. In Section 3.1, a comparison in terms of IP-level latency predictability for CGR accelerators among the main state-of-the-art HLS approaches and a novel methodology are proposed. Regarding the SW programming context, latency estimation capabilities of the approaches used for applications described as a Directed Acyclic Graph (DAG) of tasks are analyzed in Section 3.2. Section 3.3 aims at providing a more general strategy for response time estimation than what is proposed in the state of the art. This alternative explores the use of MoAs to extend the applicability of timing analysis to comples HW/SW defined CPSs by reducing complexity due to the detailed descriptions involved in the state-of-the-art strategies. Finally, Sections 3.4 summarizes problems and objectives associated with the contributions of the thesis.

3.1 Problem Definition in the CPS Hardware Context

Flexibility and performance are two highly valued properties of processing systems in many applicative domains. Modern systems include widely deployed and upgradable devices for cyber-physical applications that also communicate with cloud infrastructures. These systems must adapt to mutable conditions, while avoiding unpredictable performance degradation.

To boost performance, HPC and embedded systems designers are pushed to opt for Domain-Specific Accelerators (DSAs), built from ad-hoc hardware-coded kernels exploit-

ing parallelism for optimizing performance. Two representative examples of this trend are Amazon Web Services (AWS) and Intel Movidius. On the HPC side, AWS offers FPGAs in a cloud environment [Bar17]. On the embedded side, the Intel® Movidius™ Vision Processing Unit (VPU) is a low-power DSA used in smartphones and drones for computer vision and artificial intelligence applications.

In terms of system flexibility, hardware DSAs based on CGR architectures offer flexibility by adapting to variable application parameters. CGR architectures are traditionally composed of a mesh of PEs whose interconnections are reconfigured over time [WWC16] to offer flexibility. CGR architectures inherit from their hardware nature the capacity of executing compute-intensive kernels in an energy-efficient way. However, efficiency and flexibility do not come for free. HW design is a complex and error-prone task, leading to productivity losses, especially for heterogeneous and irregular targets. HLS approaches have been proposed to cope with these losses, obtaining design productivity gains by separating functional system verification, performed from a time-agnostic high-level language, from timed system verification, performed after automatically inferring hardware-specific code [Pel+16].

As explained in Section 2.2.2, many HLS tools are now available, such as Xilinx Vivado HLS [Xil] and Intel FPGA SDK for OpenCL [Int]. These tools are based upon imperative, C-like, languages. The algorithm to be implemented is formulated as a *sequence* of instructions operating on mutable data. This choice is motivated by the very large number of developers trained to manipulate imperative languages that have been dominating computer sciences for decades. But, from a hardware perspective, this choice presents two major drawbacks:

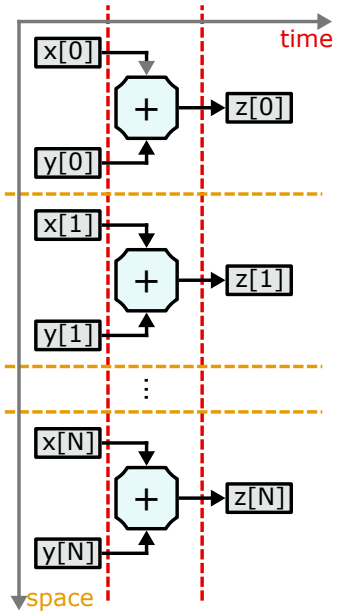
— imperative formulations generally do not distinguish iterations over time from iterations over space, which do not translate uniformly in hardware (the latter do not imply causality and can therefore be parallelized using replication). Indeed, as depicted in Figure 3.1, a high-level description (Figure 3.1A) can correspond to different time/space solutions by replicating HW blocks as enabled by using or not techniques such as unrolling (see Figures 3.1B and 3.1C respectively).

— imperative formulations implicitly rely on the concept of global memory at the implementation level (see Figure 3.2A), which in turn leads to a "bottleneck" on memory accesses [Bac07].

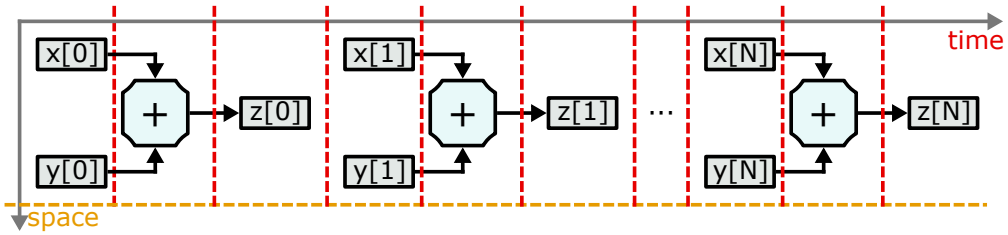
These drawbacks can be circumvented by relying upon so-called applicative or functional languages in which algorithms are described as a (mathematical) composition of side-

```
void ip(...){
    ...
    for(i=0;i<=N;i++){
        //read, compute and write
        z[i] = x[i] + y[i];
    }
    ...
}
```

(A) High-level specifications

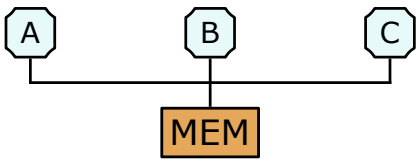


(B) With HW unit repetition

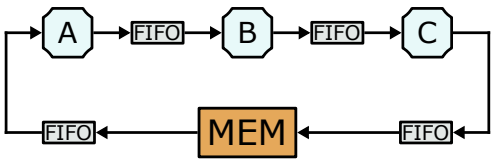


(C) Without HW unit repetition

Figure 3.1 – Example of imperative formulations in terms of iterations over time and space.



(A) Global memory approach



(B) Stream-based approach

Figure 3.2 – Comparison of stream-based and global memory approaches.

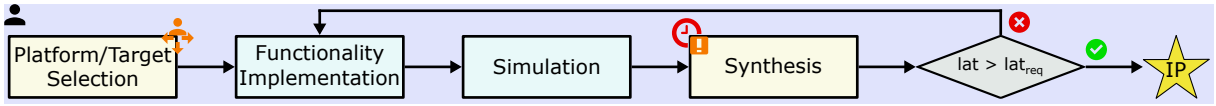
effect free functions. This approach naturally fits DF MoCs. An application is decomposed into independent processing actors, communicating with FIFOs, with no global storage or synchronization (as depicted in Figure 3.2B). This is particularly true for signal processing applications, processing data "on the fly" and which benefit significantly

from CGR architectures-based acceleration, as found in signal/image processing, media coding/compression, cryptography, video analytics, etc.

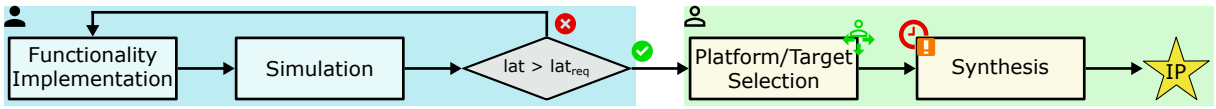
This section, introduces a novel methodology for the optimal characterization of CGR DSAs exploiting a DataFlow-Functional HLS approach as an alternative to traditional HLS tools based on imperative languages.

3.1.1 Discussion on Latency Information in CGR DSA Designs

Knowing latency of the implemented system at early design stages is an important tool for improving productivity and reducing design effort. Moreover, platform-agnostic HDL specification is challenging, as a resulting generic hardware has difficulties to compete with platform-specific code performance. However, this property enables re-use in contexts in which a wider DSE is evaluated. In this section, approaches based on providing response time estimation before and after the RTL synthesis process and depending on the platform/target selection are described.



(A) Platform-/Target-Dependent Approach with Post-Synthesis Information



(B) Platform-/Target-Agnostic Approach with Pre-Synthesis Information

Figure 3.3 – HW time verification strategies.

Platform-/Target-Dependent Approach with Post-Synthesis Information

Figure 3.3A illustrates the design methodology of the main imperative-based target-dependent HLS tools present at the state of the art. As a first step, the IP developer needs to decide the platform (ASIC or FPGA) and the final target depending on the vendor (such as Xilinx and Intel). The following phases are strictly characterized by this design choice, whether they are related to the target hardware or to the functionality (respectively depicted as light yellow and light cyan boxes). In order to implement and optimize the description of the functionality, developers have to understand how to exploit

both the available `#pragmas` (i.e. metadata added in the imperative code to orientate synthesis) and their combinations towards best performance. Nevertheless, simulation of the functional behavior provide no information about the achieved response times. Indeed, an often computationally-intensive and target-dependent process capable of extracting this knowledge is needed: the RTL synthesis. It consists of assigning HW resources to each combinatorial and sequential construct implementing the application. In addition, code refactoring may also be necessary to achieve optimal latency values. As a consequence, this type of HLS tools require a DSE to obtain the required system execution latency ($lat > lat_{req}$), iterating among implementation, simulation and synthesis. Moreover, in order to deploy a corresponding IP for a new target hardware, the functionality needs to be re-implemented according to the specifications associated with the new target. For these reasons, developers must have a thorough knowledge of the synthesizer and a considerable effort is required to obtain an optimized architecture, especially in contexts requiring different targets.

Platform-/Target-Agnostic Approach with Pre-Synthesis Information

The DFF HLS proposed in this work is designed to offer response time evaluation *before synthesis*, relying on dataflow-based tools and functional programming. *Pre-synthesis* time prediction gives developers the advantage of focusing on the application to be accelerated with a functionality-oriented approach rather than a synthesizer-based one. Moreover, when the description of the algorithm leads to platform-/target-agnostic synthesizable HDL, the development of the IP can be split into two parts associated to the application and to the hardware respectively (which can also be entrusted to different developers). As shown in Figure 3.3B, the high-level simulation of the generic specification provides response time estimation in terms of clock cycles (CCs). Thus, in order to reach the required system execution latency, the optimization of the functionality does not keep the developer waiting for the important times linked to the synthesis. Moreover, since it is relevant for different platform natures, DFF HLS can be re-used in contexts with a wide spectrum of implementations. To conclude, when compared to imperative HLS, the DFF nature of the proposed HLS provides earlier explorations, while offering system predictability and guaranteeing performance.

3.1.2 Motivation on Creating DFF HLS CPS Hardware Design

Regarding the frameworks related to the CGR design (reported in Table 2.4), to the best of our knowledge, neither [Raf+10] nor [Vor+13] present the capability to provide pre-synthesis response time estimation, while [Sau+14] handles SDF inputs only. Although the designer needs to model reconfiguration by hand, CGR DSAs can also be obtained with sole HLS. Nevertheless, state-of-the-art HLS frameworks are based on imperative languages, and most of them are also target-dependent [Nan+16].

HLS tools based on imperative languages can only provide IP response time estimates *after synthesis* when a reconfigurable system is designed. In Vivado HLS [Xil] this estimate takes the form of a range (min to max latency): the actual value of each single configuration in a reconfigurable case is obtained by running the corresponding mode over the deployed CGR accelerator. The Intel FPGA SDK for OpenCL [Int] pursues automatic system-level integration (and parallelization) of HW accelerators complying the OpenCL computing model. As a result, low-level, cycle-accurate pipeline/scheduling information, typical in other IP-based HLS tools, is sacrificed in favor of an annotated dependence graph describing the generated kernel as a combination of blocks. The estimation provided in the graph refers to the types/sizes of load/store units, stalls and latencies, but a non-trivial calculation of the IP response time is left to designers. Finally, optimization is performed by profiling kernel execution, requiring re-synthesizing an equivalent kernel, automatically instrumented with performance counters, to analyze memory access behavior (stall, occupancy, bandwidth).

In addition to relying on imperative languages, Vivado HLS and Intel FPGA SDK for OpenCL are target-dependent (Xilinx and Intel FPGA devices respectively). The Synflow Studio tool [Syn] provides a proprietary HLS framework for multi-vendor FPGAs using the Cx DF language, which does not support ASICs. To the best of our knowledge, only two academic flows support platform-agnostic DF-oriented HLS: the Orcc HDL backend tool [Sir+10] leveraging on the RVC-CAL language, and CAPH [SBA13]. CAPH, adopted in the proposed flow, has the advantage of using functional programming semantics that, contrary to the RVC-CAL language, removes imperative semantics from actors. The Orcc HDL backend is no longer supported, thus it is not possible to compare the performance of our proposed solution with [Sir+10]. Moreover, our platform-agnostic DFF-oriented HLS needs to deal with the reconfiguration management of the custom IP in order to reach high performance in a CGR DSA design. As reported in Table 2.4, the DF-based suite for CGR accelerators MDC provides maximization and control of the resource re-use, and

reconfiguration, leveraging on datapath merging techniques.

3.1.3 Objectives of this Thesis on Hardware Design Automation

The key selling feature of the proposed DFF HLS is its ability to compute early and accurate system execution latency [1] estimations *before synthesis*. Such values depend on the critical path length in the dataflow network and on the maximum number of cycles required for each actor firing. For SDF graphs, both can be computed statically. For non-SDF graphs, CAPH estimates system execution latency of a complete execution by running the cycle-accurate code generated by its SystemC backend. The response time of the CGR-based DSAs generated by the DFF flow depends on the selected mode only, corresponding to the selected stand-alone networks, since merging only adds combinatorial switching elements for the configuration of the accelerator which do not affect system execution latency. As a result, the DSA can be optimized in terms of response time *before synthesis*.

The proposed methodology is targeted for heterogeneous and irregular CGR architectures, leveraging on application specific PEs and tailoring the interconnect to minimize FIFOs. For prototyping purposes, and to compare with commercial flows, experiments target FPGA technologies and demonstrate that, thanks to its modularity and abstraction capabilities, the proposed approach guarantees latency predictability (Section 4.1.3). Moreover, the DFF approach leads to improved performance for DSAs with CGR architectures.

3.2 Problem Definition in the CPS Parallel Software Context

High performance embedded systems and CPSs now process, close to sensors, complex workloads that need to be spread over heterogeneous and specialized PEs to comply with systems timing constraints. In this context, performance measurement is crucial in achieving efficient solutions with respect to the relevant KPIs, such as throughput and latency. Latency , also called *system execution latency* or *response time*, has varied definitions in various communities. Indeed, its definition requires the notion of a unit of execution,

1. Number of clock cycles required to compute all outputs as in [Xil].

potentially indefinitely repeated on input data streams, whose lifetime in the system defines latency. Regardless of the type of applicative workload (signal processing, stream processing, batch processing, etc.), response time is determined by a succession of causal, time consuming mechanisms that are usually representable by a DAG of data-dependent tasks, representing one execution iteration. The application DAG then serves as the entry point for execution time studies [FW19]. On the HW level, heterogeneous MPSoCs are efficient solutions when executing multi-functional applications with workloads that can vary depending on timing requirements. However, when an application is parallelized and scheduled on an MPSoC, the performance of the system in terms of response time is difficult to predict and understand from application and architecture models. Indeed, latency is a highly non-linear property, affected by many software, hardware, and scheduling phenomena.

This section introduces a new methodology for modeling the system execution latency. The objective of the developed method, named PathFinder, is to extract, from an application model, a Longest-Latency Path (LLP), that is a subset of the application workload that approximates the response time. Generalizing this concept, we refer as *activity* to the share of an application workload that determines a given KPI [Pel+18]. In this context, the LLP is the application activity for system execution latency. Having such information on the application workload opens up a large set of studies, ranging from abstract Model of Architecture (MoA) design to scheduling optimizations and design space exploration.

PathFinder aims at building a model of the MPSoC workload execution that offers more insights on response time activity than the Deterministic Actor Execution Time (DAET)-based solution provided by schedulers. The following sections detail the input MoC for the method (Section 3.2.1), a motivating example showing that straightforward DAG scheduling does not properly model real execution (Section 3.2.3), a discussion on the high-level information that can be exploited from the architecture to model latency more accurately (Section 3.2.2), and the objectives of the proposed PathFinder method (Section 3.2.4). Chapter 5 will detail the pros and cons of PathFinder.

3.2.1 Application Representation

The application representation input to the PathFinder method complies to a subset of the SDF [LM87] MoC corresponding to a transformation into a DAG. In this form, production and consumption rates on each FIFO in the graph are made identical. Moreover, to this general graph, the following simplifications are applied: i) no cycle is considered

(FIFOs from one iteration of the graph to the next are ignored) and ii) it is assumed that the number of initial data packets (*delay*) in each FIFO is zero. A formal definition follows.

The DAG model is represented as a finite directed, weighted graph $G =< V, E, m >$ where:

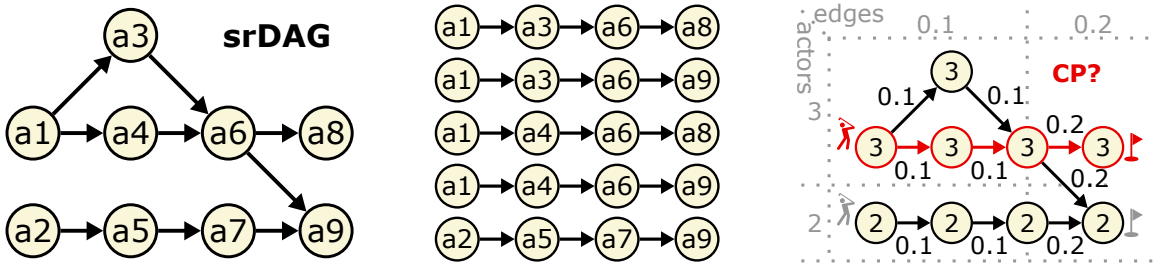
— V is the set of nodes called actors; each node represents a non-preemptive task that performs computation on one or more input data streams and produces one or more output data streams. Task execution is data driven: the task is fired as soon as input messages are available on all input edges.

— $E \subseteq V \times V$ is the edge set, representing messages between tasks.

— $m : E \to \mathbb{N}^*$ is the *message size* function with $m(e)$ representing the number of data indivisible units (called *tokens*) sent from the e source actor to the e sink actor in a message.

The absence of cycles and delays makes the consistency and liveness studies of the graph trivial and its *single-rate* nature makes all tasks executed only once per iteration of the graph. One may note that for the graph to be alive, all source actors, i.e. actors with no input edge, shall be fired at the same rate. This graph is a coordination language. It only specifies the topology of the network, but does not give any information on the internal behavior of tasks. However, each actor is associated to a code implementing its internal functionality (e.g., a C code as in Section 5.2). Moreover, information is tagged on tasks and messages to model the relative cost of their management.

Such a model is called single-rate DAG (srDAG) in the literature and has the advantage of simplicity at the cost of a lack of scalability for heavily multirate applications (e.g., see [Hsu+07]). This srDAG entry point (depicted in Figure 3.4A) is chosen because it is common to many studies in the literature (e.g., [FW19; Li+17; Now+13; Mel+15; Bon+19]) and can be generated from many applicative representations, ranging from advanced DF algorithms (such as SDF, CSDF [Bil+96] or PiSDF in our case [Des+13]) to real-time application task sets. In the following discussions, we will refer to *DAG* or *srDAG* interchangeably, considering their meanings corresponding to the definition given in this section.

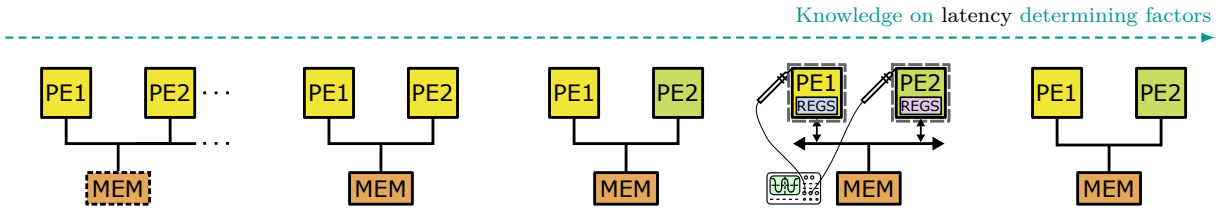


(A) Application (B) CP Candidates (C) srDAG Characterization

Figure 3.4 – Topological-based latency evaluation of a dataflow application.

3.2.2 Discussion on the Levels of Exploitable Architectural Information

The next sections list incremental levels of architecture knowledge, discussing the determining factors that can be used when modeling response time. Figure 3.5 illustrates the different levels of architectural information that can be leveraged on to predict system execution latency. The discussion starts from the steps (A, B and C) based on the information available before running the DAG on the MPSoC, such as DAET tags and architectural properties (called *a-priori information*). On the other hand, the subsequent levels (D and E) consider tags of the start/end time, obtained only after the DAG execution and named *a-posteriori information*.



(A) Inf. Hom. PEs (B) Fin. Hom. PEs (C) Fin. Het. PEs (D) Real Archi (E) PathFinder

Figure 3.5 – Latency evaluation strategies depending on the level of architecture knowledge.

A-Priori Information Level A – DAG with Single DAET Tags

Several works have analysed the theoretical latency of the DAG depending on its Critical Path (CP) [Sin07; KAL11; Pel+13]. As shown in Figure 3.4B, the DAG is divided

into task chains starting from each source node and ending to every sink node, that in this paper will be called *CP candidates*. The latency evaluation from CP analysis is often performed by using heuristic approaches based on path exploration and task characterizations. For each CP candidate, DAG nodes are tagged with timing weights to compose its specific latency. Thus, the CP is characterized by the following assumptions: i) it consists of a task chain that maximizes the sum of contributions, and ii) the CP determines the latency of the whole application (as shown in Figure 3.4C). In this context, since no information about the architecture is considered, CP is considered to independently run on a system with unbounded (or large enough) memory and infinite (or as many as necessary) number of homogeneous PEs, as depicted in Figure 3.5A).

A-Priori Information Level B – DAG with Single DAET Tags and Architecture with a Specified Number of PEs

With a known number of homogeneous PEs and a bounded memory (see Figure 3.5B), a heuristic-based scheduling/mapping strategy can be applied in order to achieve an optimized solution in terms of timing performance [Sin+13]. This procedure leads to a more realistic model of execution in which paths do not need to be analyzed in the latency representation (often depicted with Gantt charts). As tasks share PEs and messages share memory, contentions appear on architectural resources among tasks or communication operations. These contentions lead to interferences in DAG execution, which in turn increase latency and, if not predicted, reduce latency predictability.

A-Priori Information Level C – Architecture with a Specified Number of PEs of Different Types, and DAG tagged with one DAET per PEs Type

A more accurate solution in terms of the DAG execution time can be achieved by distinguishing the types of the PEs in the target system (see Figure 3.5C). Nevertheless, with respect to the homogeneous case, this implies to provide latency prediction with the characterization of: i) the tasks for each type of PE, and ii) the elements of the hardware infrastructure. To do so, a DSE consisting of a statistical evaluation of the target-dependent costs associated to computation and communication may be required.

A-Posteriori Information Level D (Trace) – DAG with Start and End Time Tags and Real Architecture

Figure 3.5D highlights the differences between i) the model of the system interconnects and the bus system linking multiple PEs, and ii) the abstract memory and its actual hierarchical structure. This case corresponds to an execution trace, extracting from code executions the start and end times of all tasks. It can be used as the reference Gantt chart to be looked for by models. At this post-execution stage, system execution latency is perfectly known. However, the complexity of the monitoring procedure increases when not only task start and end dates are observed but also communication related contention. Indeed, several factors, such as task synchronization and data movements, combine during the management of the DAG messages, and this requires complex monitoring in order to obtain a detailed information. Regarding explainability, such a trace does not provide a full knowledge on the execution, as only start and end times of tasks are known but the causes of potential time variations and instants of data transfers and synchronizations remain unknown.

A-Posteriori Information Level E – DAG with Start and End Time Tags, Full Scheduling Information and Architecture with a Specified Number of PEs

We propose PathFinder as a method for creating this new level of architectural information from system monitoring (Figure 3.5E). This level extracts the LLP as the subset of tasks causing latency. In order to explain the monitored latency, PathFinder analyzes the causes for each LLP task. Indeed, since the previous levels (A,B,C) rely on models with average task behaviors, a degree of uncertainty exists that, due to the strongly non-linear nature of latency, results in large modeling errors. This error is further discussed in the next section. In order to obtain an insight of the execution, the examination can be based only on the start and end times of the tasks. In particular, if the task are scheduled non-preemptively, their latency contribution can be identified using performance monitors. In such a case, we can benefit from the advantages of a DAG-based model and make latency predictions closer to reality than with levels (A,B,C). In particular, when task contributions dominate the cost due to communication and synchronization, this fact can be exploited to simplify modelling. As the primary goal of PathFinder is to target shared memory heterogenerous MPSoCs that constitute a largely deployed set of HW platform, most latency cost of communication is considered embedded into tasks, as tasks access

memory and activate cache write-backs while computing. Our experimental results show that this hypothesis is acceptable on our test platform. For this reason, messages are not scheduled and there is no communication-related line in the displayed Gantt charts.

3.2.3 Motivating Example: Latency Evaluation of a DAET Tagged DAG Scheduling

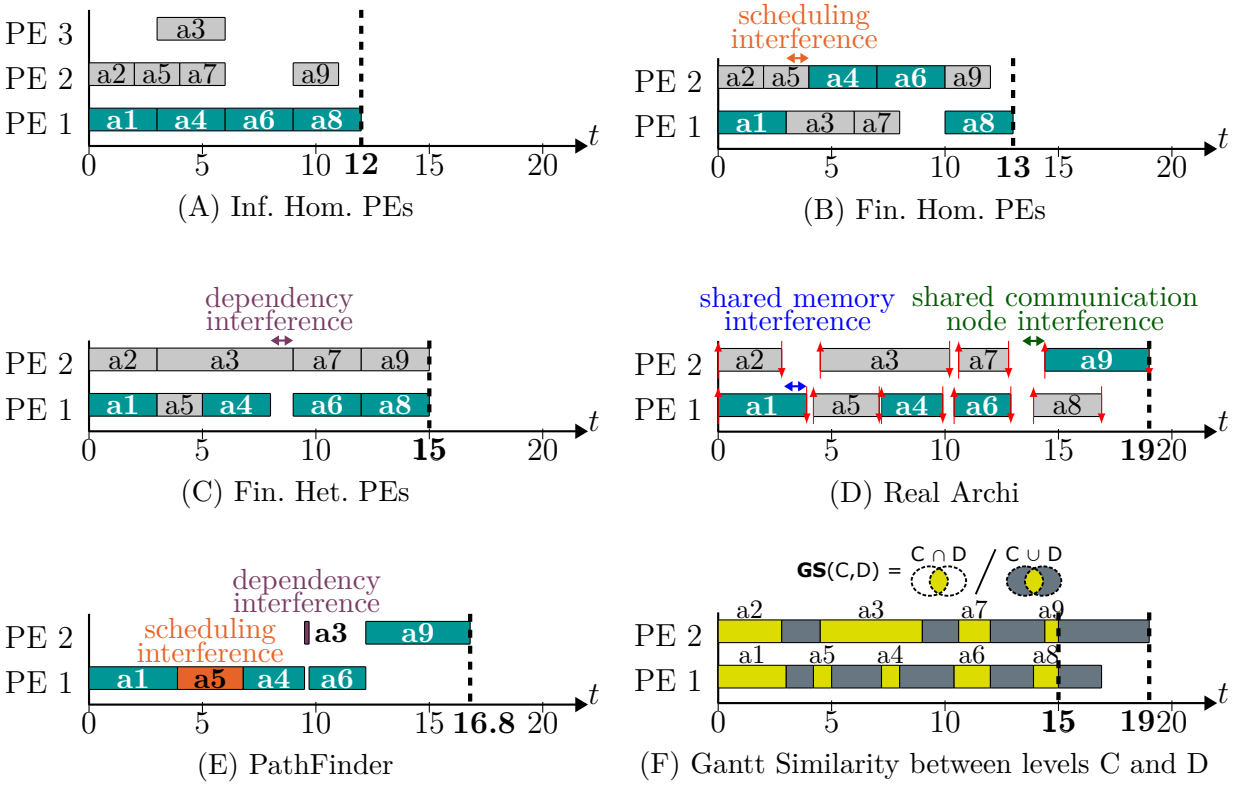


Figure 3.6 – Gantt chart with tasks execution times for each level of architecture knowledge depicted in Figure 3.5.

For each step in Figure 3.5, possible Gantt charts are shown in Figure 3.6 (where the CP actors are depicted with teal color). Moreover, they are based on the actor and edge characterization as in Figure 3.4C and its double values for the PE1 and PE2 clusters respectively. In this example, to accentuate the possible differences between real execution (Level D) and its models (Levels A, B, C, E), the communication costs are considered doubled in case of token transfer among different clusters.

Although level C requires an advanced a-priori knowledge on tasks execution times and tasks causality, it provides low accuracy estimations and improper explainability of

the real execution. For evaluating this mismatch, we translate the concept of *Jaccard index* [Rez+19], also called Intersection over Union (IoU), for comparing Gantt charts. We name this metric Gantt Similarity (GS), and compute it by applying the method of IoU to the context of parallel processing. This score between 0 and 1 indicates the degree of similarity between two Gantt charts X and Y:

$$GS(X,Y) = \sum_{i=0}^{T} [IoU(X_i, Y_i)] \cdot w_i = \sum_{i=0}^{T} \left[\frac{area(X_i \cap Y_i)}{area(X_i \cup Y_i)} \right] \cdot \frac{area(Y_i)}{area(Y)} \tag{3.1}$$

where T is the number of actors present in the srDAG, and X_i and Y_i represent the boxes associated to the execution times of the i-th task in X and Y respectively. X_i and Y_i are represented as rectangles with fixed arbitrary height. Since tasks affect execution with different weights, w_i takes into account this aspect, considering the ratio between the areas of the i-th task and of all the tasks present in a reference Gantt chart (Y).

With respect to the example shown in Figure 3.6, a comparison between the Gantt charts of the level C and D leads to a GS of 0.43, i.e. a similarity of 43% (see Figure 3.6F). In reality in an MPSoC, the mismatch can be much higher, as shown in Section 5.2, due to variations in tasks processing times and communications. For this reason, PathFinder relies first on an a-posteriori execution information (i.e. task execution times are known, GS = 1), with the objective to extend PathFinder results to a-priori latency prediction. The latency difference between Figures 3.6E and 3.6D, and the remaining uncertainty in PathFinder response time prediction, come from 1) communication and synchronization times that are currently non-modelled and 2) potentially missed interferences that should have entered LLP. Indeed, the LLP is computed by heuristics and not by exact algorithms. However, we show in Section 5.2 that under these hypothesis, PathFinder is capable of modelling the LLP with high accuracy, giving insights on latency-causing task in the application.

3.2.4 Objectives on Software Latency Predictability

PathFinder aims at analyzing the determining factors that lead to the prediction gap between Gantts C and D. The proposed strategy relies on an application-centric information: the application Critical Path (CP). In particular, by decomposing the list of elements determining latency into CP and interferences, we propose the concept of Longest-Latency Path as an equivalent to CP in levels B through E. Indeed, interferences are introduced

by different phenomena, mainly: i) sharing of a finite number of hardware resources (processor, memory, interconnects, peripherals, etc); ii) the nature of the application itself (task heterogeneity, scenario-based diversity in the actor characterization, task relations, etc); iii) the random access to the memory by tasks that prevents complete determinism [HP18].

In level A, no interference arises, since CP can run independently. From B to D, the main types of interference that can be observed are associated to:

— *scheduling*: when a CP actor execution is delayed due to other tasks that precede it in scheduling are mapped in the same PE (in Figure 3.6B, a4 cannot be executed since a5 is still running);

— *dependency*: when the execution of a CP instance has to wait for tokens coming from other tasks that are not present in the CP (as shown in Figure 3.6C, a6 is waiting for a token from a3);

— *shared memory*: that lengthens a CP task execution, since simultaneous memory accesses are limited (as depicted in Figure 3.6D, a1 is waiting for the access to the memory);

— *shared communication node*: if a CP edge execution is prolonged due to the traffic in the links between PEs and memory (still in Figure 3.6D, a7 and a6 are using the communication node at the same time).

3.3 Problem Definition in the Hybrid Hardware-Software Context

In Sections 3.1 and 3.2, discussed objectives aimed at estimating response time of functionalities implemented as a CGR accelerator, and DAG of software tasks, have been described respectively. Although these two aspects have been treated separately, their combination is often desired in contexts where reconfiguration or adaptation is required. However, scheduling and mapping such functionalities requires to generalize the strategy used in estimating system execution latency. As seen in Chapter 2, the characterization of a functionality depends on the MoC selected at design time. Moreover, the HW system affects the response time according to the use of the PEs, memory and interconnects. For this reason, the architectural elements are often modelled in detail. Nevertheless, this leads to a high degree of complexity that reduces the applicability domain of the timing

analyses. In particular, this fact can have a strong impact on the support of scheduling and mapping strategies, especially when parametric applications are mapped on heterogeneous systems involving general-purpose and CGR PEs. In order to extend the feasibility of the response time evaluations and efficiently exploit them for adaptation purposes, a more simple activity that determines the latency costs needs to be considered. However, finding a proper solution in this context represents an open issue for many years.

In addition, although several tools capable of handling runtime adaptation have been proposed in literature, most of them concentrate only on SW management [AG17; Gau+13; Ger+16; HPB17; Hor+09; QP16; RBK16; YPB15; Zha+15]. HW runtime adaptation is managed in [TB04], in which data routing in a specific HoneyComb processor array hardware has been considered, and in [Bra+18], in which, besides the SW runtime handling, HW tasks can be implemented in FPGA devices. However, among the proposed frameworks only a few are open-source and effectively available [Bra+18; Gau+13; Ger+16; RBK16]. Among these, [Gau+13] and [RBK16] are HPC management systems that place themselves over large-scale facilities composed of multiple CPUs and GPUs. In [Ger+16], the framework does not consider HW acceleration, which is today compulsory in most High Performance Embedded Computing systems found in CPSs, such as embedded video processing systems, embedded deep learning, telecommunication and computer vision systems [Wol14]. Although the framework presented in [Bra+18] can handle HW tasks, these are specific to Intel FPGA devices. Therefore, a framework capable of providing runtime management of reconfigurable functionalities mapped on systems with CGR accelerators and general-purpose PEs is desirable.

3.3.1 Motivation on Latency Estimation for Hardware-Software Design Automation

In order to determine the response time of a parametric functionality, new approaches that carefully trade-off between accuracy and complexity need to be considered. In particular, the idea is to evaluate a strategy that favors a reduction of complexity at the expense of a controlled degradation of accuracy. Since the reconfiguration and adaptation requirements must be met, this new approach must nevertheless generate a model with high fidelity, i.e. capacity to feed correct decisions. For this purpose, a fidelity calculation method and new approach with high fidelity properties in KPI estimation for parametric applications performed on MPSoCs are introduced below.

Fidelity Evaluation

A useful metric for measuring the effectiveness of an estimator is a Fidelity Metric (FM). Fidelity does not measure the absolute accuracy of a model, but rather its capacity to correctly order the set of its outputs. It can be defined through the correlation coefficients of Spearman's rank ρ and Kendall's tau τ [BB00; JIP10] as follows:

$$FM_\rho = 1 - \frac{2 \cdot \sum_{i=1}^{n} r_i^2}{\frac{n(n^2-1)}{3}} = 1 - \frac{2 \cdot \sum_{i=1}^{n} (P_i^{r,a} - P_i^{r,e})^2}{\frac{n(n^2-1)}{3}} \tag{3.2}$$

$$FM_\tau = \frac{n_c - n_d}{\frac{1}{2}n(n-1)} \tag{3.3}$$

where:

— n is the dataset size of the *actual* and *estimated* points (P^a and P^e respectively);

— $r_i = P_i^{r,a} - P_i^{r,e}$ corresponds to the difference of the ranks (considered in increasing order) associated with the i-th pair of points (P_i^a, P_i^e);

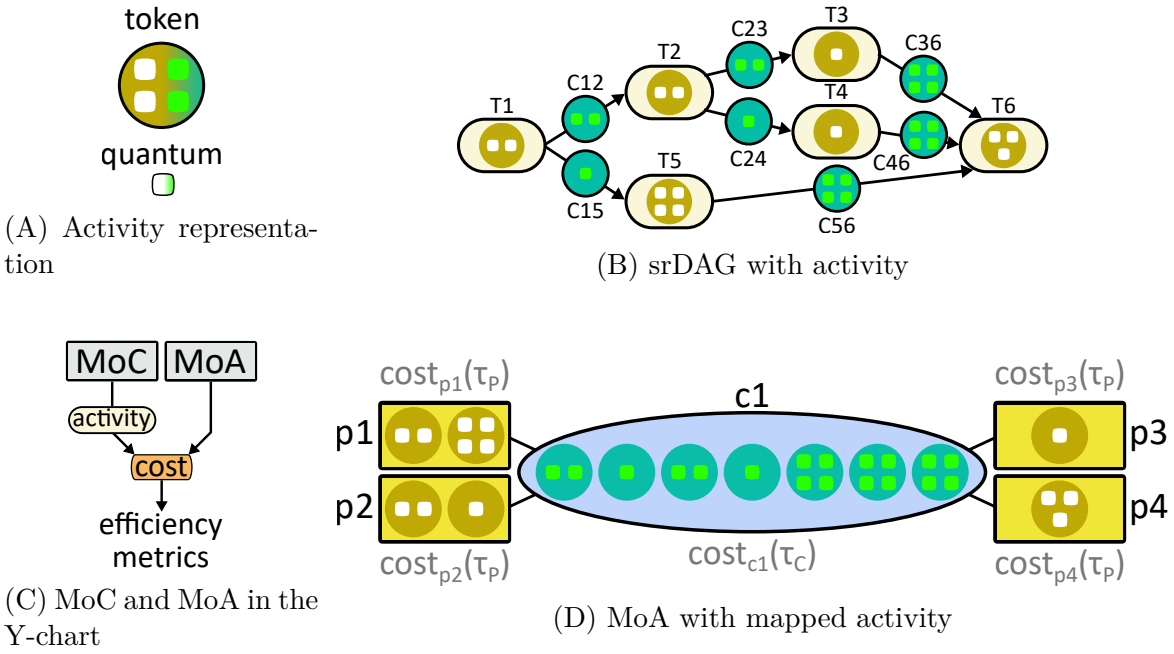
— n_c (n_d) is the number of concordant (discordant) pairs $< P_i^a, P_j^a >$ and $< P_i^e, P_j^e >$ with $i < j$, such as $sign(P_j^a - P_i^a) = sign(P_j^e - P_i^e)$ ($sign(P_j^a - P_i^a) = -sign(P_j^e - P_i^e)$) where $sign(x) = \{-1 : x < 0; 0 : x = 0; 1 : x > 0\}$,

— $-1 \leq FM_{\rho,\tau} \leq 1$, where the value 1 ($-1$) corresponds to datasets perfectly correlated (uncorrelated).

Model of Architecture (MoA)

Recently, a new architecture modeling approach based on the concepts of abstraction and modularity has been proposed [Pel+18]. Through the introduction of an MoA, the cost related to the activity in terms of a specific KPI can be defined. In this context, application *activity* represents the amount of processing and communication to be carried out in order to execute the functionality. Activity consists of a number of packets (the tokens, τ) in turn composed of smaller units (the quanta, q) (see Figure 3.7A). Tokens and quanta are associated with the processing ($\tau_P \in T_P$ and q_P) or to the communication ($\tau_C \in T_C$ and q_C), as shown in Figure 3.7B for the tasks (such as T1) and their links (e.g. C12). Intuitively, tokens correspond to units of either processing or communication, and quanta represent the unitary brick (inseparable unit) of either computation or communication.

Deriving from the Y-chart design method, the calculation of the activity depends on the MoC considered to describe the high-level application (see Figure 3.7C). Indeed, the Y-chart approach [Kie+97] consists of characterizing the development of application and architecture separately. The contribution of the activity (derived from the MoC) mapped on an MoA produces a cost in terms of the selected KPI, which is defined with respect to *cost functions* associated with each architectural node (as depicted in Figure 3.7D). In the case of energy estimation, the degree of fidelity has been demonstrated highly promising (FM_τ equal to 0.86 and 0.93 for heterogeneous systems based on CPUs and GPUs respectively). Nevertheless, studies on MoAs [Pel+18; Pay+19] are focused on energy, which, differently from system execution latency, presents additive properties by nature. This fact made it possible to use a linear characterization of the cost functions that compose the MoA, called Linear System-Level Architecture Model (LSLA). On the other hand, estimating response time may lead to nonlinear MoAs depending on the activity properties. However, since representing a model in linear terms offers a reduction in complexity and the wide set of linear algebra analysis functions, choosing an activity aimed at preserving this property is highly convenient.



(A) Activity representation

(B) srDAG with activity

(C) MoC and MoA in the Y-chart

(D) MoA with mapped activity

Figure 3.7 – Overview of the MoA approach.

3.3.2 Objectives on Latency Evaluation Hardware-Software Hybrid Systems

As will be shown in Chapter 5, PathFinder provides estimations of response time for DAG-based applications with different scenarios and scheduled and mapped on a determined number of PEs. PathFinder leverages on the notion of LLP that pinpoints the relevant activity for response time analysis. Therefore, LLP can properly represent the activity required to build an MoA in timing context. As in [Pel+18; Pay+19], the idea is to consider an activity that takes a the form of a set of tokens, although they derive from a DAG of tokens where edges model the causality between tokens. In particular, the proposed contribution aims at using the Linear System-Level Architecture Model MoA for the system execution latency based on the PathFinder (PF) analysis and at assessing such MoA for parametric applications on a MPSoC. For the measurement of the fidelity, (3.2) and (3.3) are involved in the evaluation.

With respect to a framework that can exploit the MoA support for the reconfiguration and adaptation strategies, a possible toolchain is introduced in Section 4.2 in order to design parametric applications mapped on general-purpose and CGR PEs. The idea is to use system execution latency cost of different mappings in order to offer a support to re-schedule and re-map the application upon HW/SW heterogeneous systems depending on user needs, environment and state of the architectural elements.

3.4 Chapter Remarks

In this chapter, a focus has been put on the definition of problems addressed by the thesis contributions. The general issue concerns the estimation of response time for parametric signal processing functionalities implemented on hardware and software hybrid systems. As a starting point, a new methodology capable of dealing with IP-level estimation for CGR accelerators has been discussed in comparison with the strategy used in the state-of-the-art toolchains. A design flow and an assessment of the proposed methodology are presented in Section 4.1.

The problem of estimating the response time is also of interest for SW applications, since the objective is to involve these accelerators in a system with general-purpose resources. Focusing on functionalities described as a DAG, difficulties in understanding factors that determine system execution latency through schedulers based on DAETs have

been discussed. In order to provide explainable and accurate estimates, PathFinder has been introduced as a new strategy. In Chapter 5, details on PathFinder with an assessment involving different use cases are proposed.

Since efficiently managing reconfiguration and adaptation of parametric applications on heterogeneous systems may lead to an explosion of the complexity in evaluating response times, simplified analyses are required. The use of MoAs has been discussed as a promising trade-off aimed at reducing complexity in timing analyses, especially when reconfiguration and/or adaptation are required. Indeed, this may enable linear evaluation of costs associated with an activity in terms of response time. Nevertheless, MoAs have not been assessed with respect to timing metrics yet. In Chapter 6, building, training and a first assessment of an MoA for response time are presented. In addition, when CGR acceleration is combined with SW task execution, a proper toolchain needs to be used. In Section 4.2, a framework based on dataflow tools is described and applied to different use cases.

Chapter 4

Contribution 1: On Building a Design Toolchain for Flexible yet Predictable HW/SW CPSs

Our objective in this chapter is to propose a HW-SW co-design flow, for signal processing applications, represented as parameterized dataflow, that outperforms state of the art in terms of latency predicability. Next contribution chapters will successively study a method to understand the applicative features causing system execution latency, called PathFinder, and an MoA unsing this activity to early estimate response time. For that we compose state-of-the-art tools in the context of the H2020 CERBERO project, combining their best HW and SW features. In particular, we combine dataflow tools and functional programming compiler. In this chapter, two DF-based design flows are combined. The objective of the study is to derive a strategy with which to manage flexible HW/SW systems depending on the system execution latency.

At first, the focus is put on reaching response time predictability for the HW components. As explained in Section 3.1, several limitations exist in the development of CGR DSAs. In order to provide an alternative that reduces these, the DFF HLS has been built (Section 4.1). This novel methodology assembles and characterizes CGR accelerators based on dataflow and functional programming principles, capable of addressing design productivity issues for CGR DSAs. The objectives have been obtained thanks to the integration of the DF-based HLS CAPH tool for streaming applications (Section 2.4.1) and the dataflow-to-hardware MDC design suite for CGR accelerators (Section 2.4.2). The main advantage of the proposed methodology is the accurate IP-level latency predictability, improving Design Space Exploration when compared to state-of-the-art HLS.

As a second step, in order to support time adaptation in HW/SW heterogeneous systems exploiting multi-core architectures and CGR DSAs, a combination of MDC and the SPIDER software runtime manager (Section 2.4.4) has been proposed (Section 4.2).

This tool integration enables the runtime handling of CGR DSAs generated by MDC. For these reasons, two Proof of Concepts (PoCs) are proposed by using the DFF HLS and by considering only MDC in Sections 4.2.2 and 4.2.3 respectively.

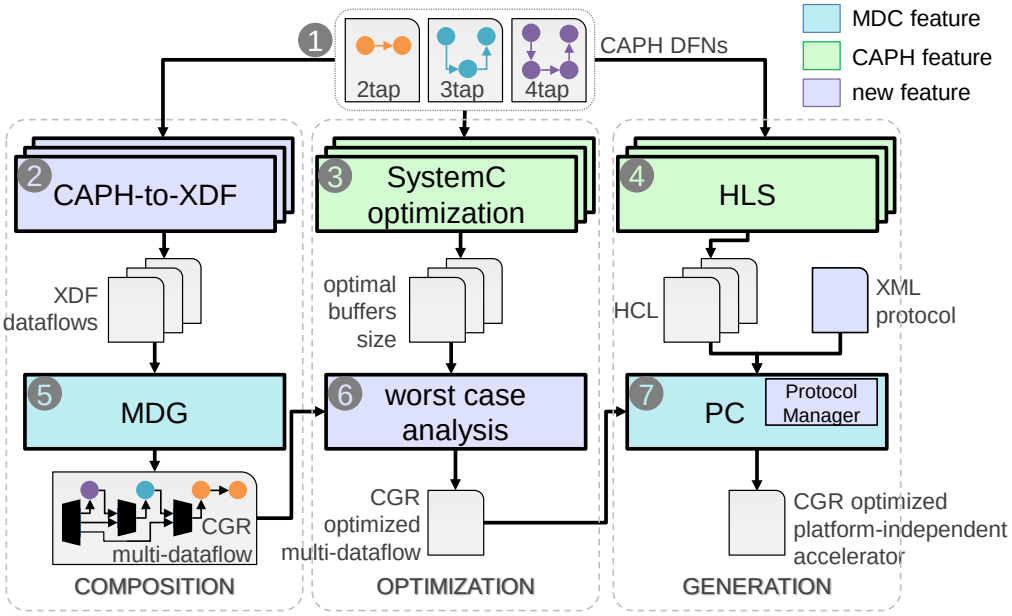
4.1 Predicting Latency in CGR DSAs

4.1.1 DataFlow-Functional High-Level Synthesis

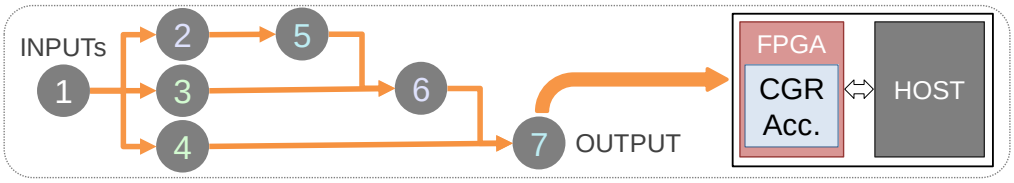
The proposed framework [Rub+19] is built on two pre-existing tools: the CAPH compiler and the MDC toolset. MDC provides N:1 DFNs mapping on a CGR DSA, but it does not support PE synthesis. CAPH provides 1:1 DFN PEs synthesis, but no support for reconfiguration. Based on the described complementaries, this work proposes a fully automated toolchain, integrating MDC and CAPH, for specifying and deploying CGR DSAs. The toolchain architecture is depicted in Figure 4.1. The proposed DFF HLS flow consists of three main phases (see Figure 4.1A):

1. *Composition* - Model-to-model compilation performed by the Multi-Dataflow Generator (MDG) component of MDC, taking as inputs generic (including CAPH) DFNs. This phase outputs a high-level multi-dataflow DFN of the DSA.
2. *Optimization* - Optimal sizing of the actor-connecting FIFOs. Optimization starts from an estimation produced by the CAPH Compiler SystemC backend, which is adapted to the multi-functional DFN case by worst case analysis.
3. *Generation* - Deployment of the CGR DSA. The MDC PC component outputs a top-level HDL module (in the Verilog language), corresponding to the optimized multi-functional DFN, using the CAPH generated Hardware Component Library (HCL) actors. With respect to writing HDL, writing CAPH DF hardware is fully synchronous by design and data-triggered processing is automatically generated.

Figure 4.1B summarizes the flow steps. (1) Users are required to provide the input set of specifications (N different DFNs) to be accelerated using the CAPH language. Starting from these inputs, three parallel steps take place to i) make CAPH DFNs compliant with MDC through the CAPH-to-XDF parser (2), ii) optimize sizing of the buffer connecting DF actors (3), and iii) generate the target-independent HCL (4). Three more steps come in succession. The N XDF networks produced by (2) are merged in a multi-dataflow network (5), which is optimized (6) using the worst-case analysis resulting from (3). Finally, the CGR platform-independent DSA is deployed in (7). During the merging process (5),



(A) Components of the flow



(B) Execution sequence

Figure 4.1 – Proposed DFF HLS.

common actors are identified and connections minimization is ensured. In the CGR DSA, combinatorial switching elements, also called Switching Boxes (SBs), are used to access shared modules, this may affect frequency but not latency.

In summary, with respect to the context of CGR DSAs, the main benefits/features of the proposed integrated flow are:

1. *Custom PE Generation* - HW generation considers heterogeneous, HLS-generated PEs for each DF actor, favoring flexibility with respect to [Raf+10].

2. *Reconfigurability Management* - DF-based mechanism maximizes and controls resource re-use, leveraging on datapath merging techniques. Moreover, reconfiguration management is provided by MDC.

3. *Predictability* - Modular specifications facilitate the predictability of system properties. *Before synthesis*, latency estimations can be carried out on the basis of the pre-processed CAPH DFNs. Since the MDC datapath merging process inserts only combinatorial switching elements [Pal+12], DF performances such as processing latency are not altered (at least in CC but frequency can suffer).
4. *Target Independence and Availability* - Both MDC and CAPH are platform-agnostic and open source.
5. *Code Readability* - MDC and CAPH preserve the correspondence among DF actors and hardware PEs. This is important for example in case of post-HLS enhancements. Vivado HLS acts differently since it assigns IPs to functional units (see Figure 4.2).

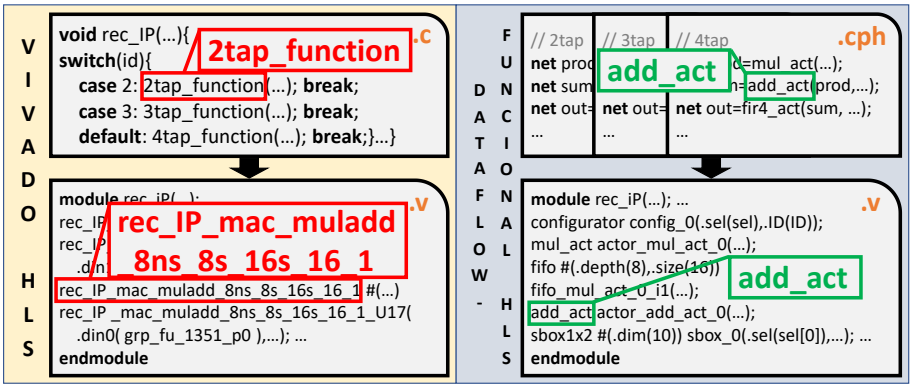


Figure 4.2 – Proposed DFF HLS vs. Vivado HLS: written and generated codes.

4.1.2 Use-case Application

Most video capable CPSs embed compression mechanisms in the domain of video coding. High Efficiency Video Coding (HEVC) is a recent format provided by the main players in developing video compression standards: ISO/IEC MPEG and ITU-T VCEG. Since HEVC offers a data compression gain of up to 50% compared to previous standards (such as Advanced Video Coding (AVC)), this codec has been successfully embedded in video systems. Nevertheless, the increasing demand for higher resolutions and frame rates led to a higher complexity of the HEVC encoding and decoding hardware. One of the most costly algorithm in HEVC decoder computation is due to the motion compensation [Nog16]. This technique, implemented with image interpolation filters, enables the data compression to exploit the temporal redundancy present in video streams. In the HEVC

standard, two 1D filters associated to the horizontal and vertical directions perform the interpolation. Finite impulse response filters with N taps constitute the filtering blocks. The considered HW implementation [Sau+17] of their structure consists of the following elements corresponding to sequential stages of the architecture:

— a pipelined channel with depth N: this hardware performs the interpolation for the incoming pixel associated to the horizontal direction;

— $N-1$ FIFO memories: in which the outputs of the pipelined channel are stored without applying any clipping;

— N parallel multipliers: the product between coefficients and interpolated data (current and previous $N-1$ rows in the FIFOs) is computed with these blocks;

— an adder tree: this completes the vertical interpolation;

— a right shifter: the output of the adder tree is shifted to store the proper value;

— a clipping block: after shifting, the result is clipped to have the original bit length.

In order to reduce filtering complexity, approximated filters with a low length N equal to 8, 5 and 3 respectively have been chosen in the proposed use case [Nog16]. The CGR DSA implementing all these configurations has been obtained for 1D filter and its 2D version (as a composition of two 1D stages) by using the design flow described in Section 4.1.1. Table 4.1 reports the number of elements of the DF specifications associated to the fixed (with 3, 5 and 8 taps) and reconfigurable (R, with runtime filter switching) designs.

Tap	**1D Filter**		**2D Filter**	
	No. of Actors	No. of Edges	No. of Actors	No. of Edges
3	11	17	19	30
5	17	27	31	50
8	26	42	49	80
R	30	45	55	86

Table 4.1 – Number of actors and edges in the implemented configurations.

4.1.3 Experimental Results

This section compares results obtained with three design flows: i) the proposed DFF HLS, using Vivado v2015.2 and Quartus v17.1 to synthesize the CGR DSAs; ii) Vivado HLS v2015.2; and iii) Intel FPGA SDK for OpenCL v17.0. Target devices are Xilinx

Virtex 7 and Intel Cyclone V FPGAs. In the case of Xilinx/Intel tools, a moderate effort has been put into code refactoring for optimization using shift register inference, line buffers, resource reuse when possible, and pragmas (loop unrolling and pipeline). For this reason we show results for two reconfigurable designs: baseline (R) and optimized (R*). A comparison among the different flows is proposed in terms of resource utilization and early estimation of the response time. By early estimation we refer to an estimate obtained before the synthesis process.

DFF HLS vs. Vivado IP-Level HLS

Table 4.2 shows that for standard implementations (indicated by 3/5/8/R) the DFF flow achieves better latency results. The very large latencies obtained with Vivado HLS are explained by the default area-driven synthesis (which required adding `#pragma` and code refactoring). Moreover, as stated in Sect. 3.1.1, the proposed flow preserves the latency values of the reconfigurable IPs, meaning that merging does not alter response time in terms of CCs as opposed to other synthesizers. Moreover, estimations before synthesis are equal to actual post-synthesis results. With respect to the optimized reconfigurable versions (R*), only for the 1D filter a lower use of resources (see Figure 4.3) and latency has been obtained using Vivado HLS (-72.0% of registers, REG; -76.1% of logic elements, LOGIC; -10.5% for minimum latency and +14.8% for maximum latency). In the 2D filtering case, DFF HLS performances are superior. Latency results are more than 3 times larger in the best Vivado HLS case, and again the latency is (i) not preserved and (ii) highly variable in the reconfigurable case.

DFF HLS vs. OpenCL System-Level HLS

As in the Vivado HLS case, a significant effort on code refactoring is needed for optimization, which is directed by an optimization report offered at early design phases (it contains highly inaccurate resource and system-level pipeline latency/stalls estimations, rather than individual kernel latency values). Hence, OpenCL does not provide a method to accurately calculate kernel latency and have insights into the generated datapath; therefore, no response time results are reported in Table 4.2. For resource usage, results refer to the kernel instantiation (See Figure 4.4). The support for the OpenCL HLS model even at kernel level (automatic pipeline to support various threads on the fly, memory accesses optimization, replications, etc.) introduces significant resource overheads.

Filter	Tap	**DFF HLS** [CCs estimate]	**Vivado HLS** [CCs estimate]
1D	3	76	1615
	5	78	2403
	8	81	3479
	R	76,78,81	2171-5771
	R*	—	68-69
2D	3	4365	504557
	5	4369	1135201
	8	4375	2409163
	R	4365,4369,4375	543782-3681252
	R*	—	15106-38999

Table 4.2 – DFF HLS vs. Xilinx Vivado HLS on Xilinx target FPGA (XC7VX485T) in terms of latency (represented in clock cycles).

4.1.4 Section Remarks

The results suggest that DFF HLS is a promising alternative to classical imperative HLS to build flexible and predictable CGR DSAs. Competitive resource usage are achieved with respect to commercial HLS methods, while allowing pre-synthesis and exact datapath latency predictions. By trying to ease hardware design to software developers, mainstream HLS tools have probably missed a key point in HW design that is critical to embedded systems: having precise information on the generated datapaths to better optimize HW accelerators through improved Design Space Exploration. We demonstrated this issue on latency, proposing an alternative design flow that overcomes traditional HLS tools under this aspect. DFF HLS is able to tackle both issues: raising the abstraction level with respect to RTL writing timeless code and letting the tool generate data-triggered code, while keeping low-level performance estimates available for further HW tuning. However, the current assessment has been focused on a single accelerator. In order to deal with the wider context of HW/SW systems, a novel design flow has been considered, as described in the following Section 4.2.

4.2 Adding Software in the Loop

In this Section, an open-source adaptive management system [Rub18] is proposed, portable over several embedded software systems and heterogeneous CGR DSAs. In or-

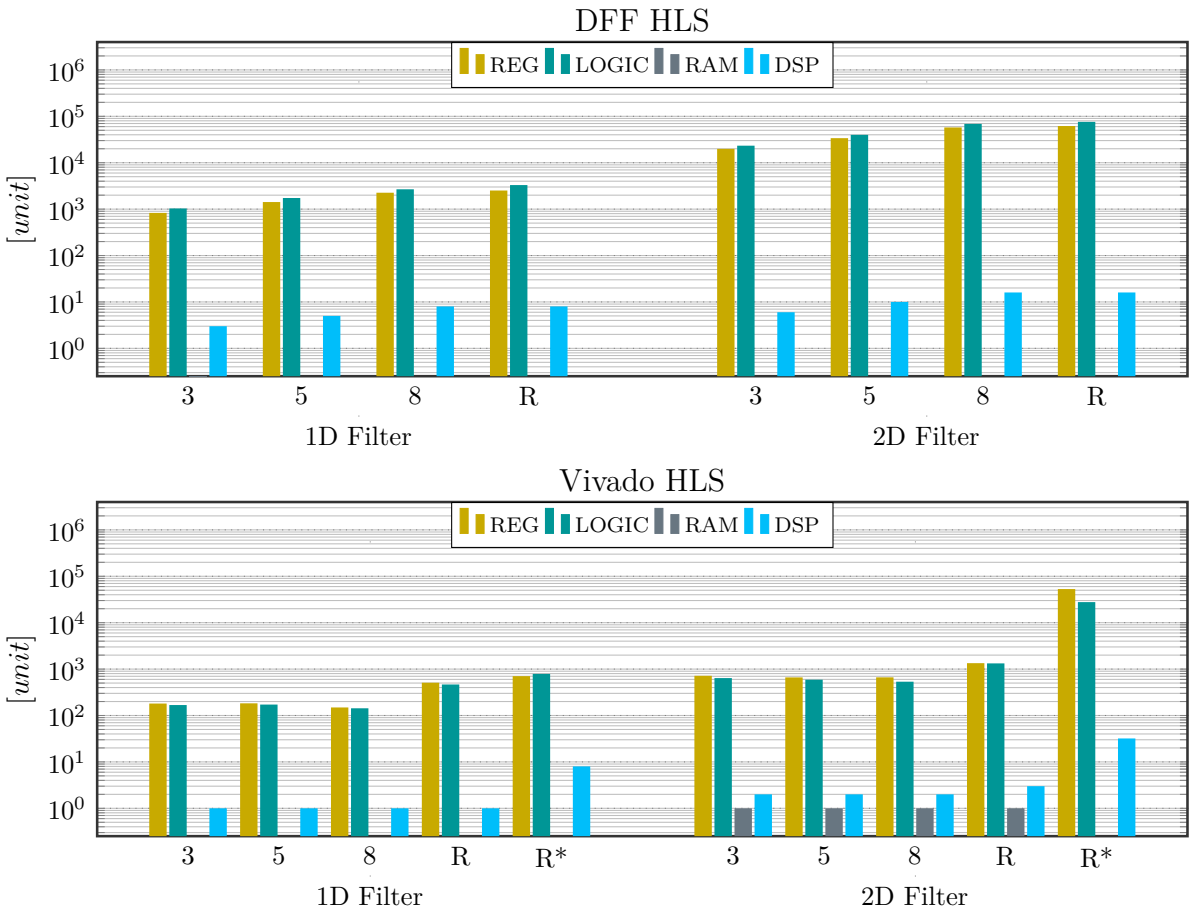


Figure 4.3 – DFF HLS vs. Xilinx Vivado HLS on Xilinx target FPGA (XC7VX485T) in terms of Xilinx resources: REG=FF, LOGIC=LUT, RAM=BRAM, DSP=DSP. Data are shown on a logarithmic scale with base of 10.

der to combine these PEs with multi-core architectures, the integration activity between the SPIDER DF-based runtime manager (see Section 2.4.4) and the MDC dataflow-to-hardware suite (see Section 2.4.2) has been conducted. SPIDER and MDC show complementary characteristics that motivate their integration. Both tools are based on a DF MoC that can be used to separate temporal and functional problems in HW design. Moreover, modularity of the dataflow representations favors a natural splitting of the computation into different blocks, making it possible to automatically map them onto heterogeneous PEs. SPIDER provides SW scheduling and memory management at runtime for multi-core architectures. However, SPIDER supported processing elements do not include reconfigurable HW blocks and adaptation is based on an a priori knowledge

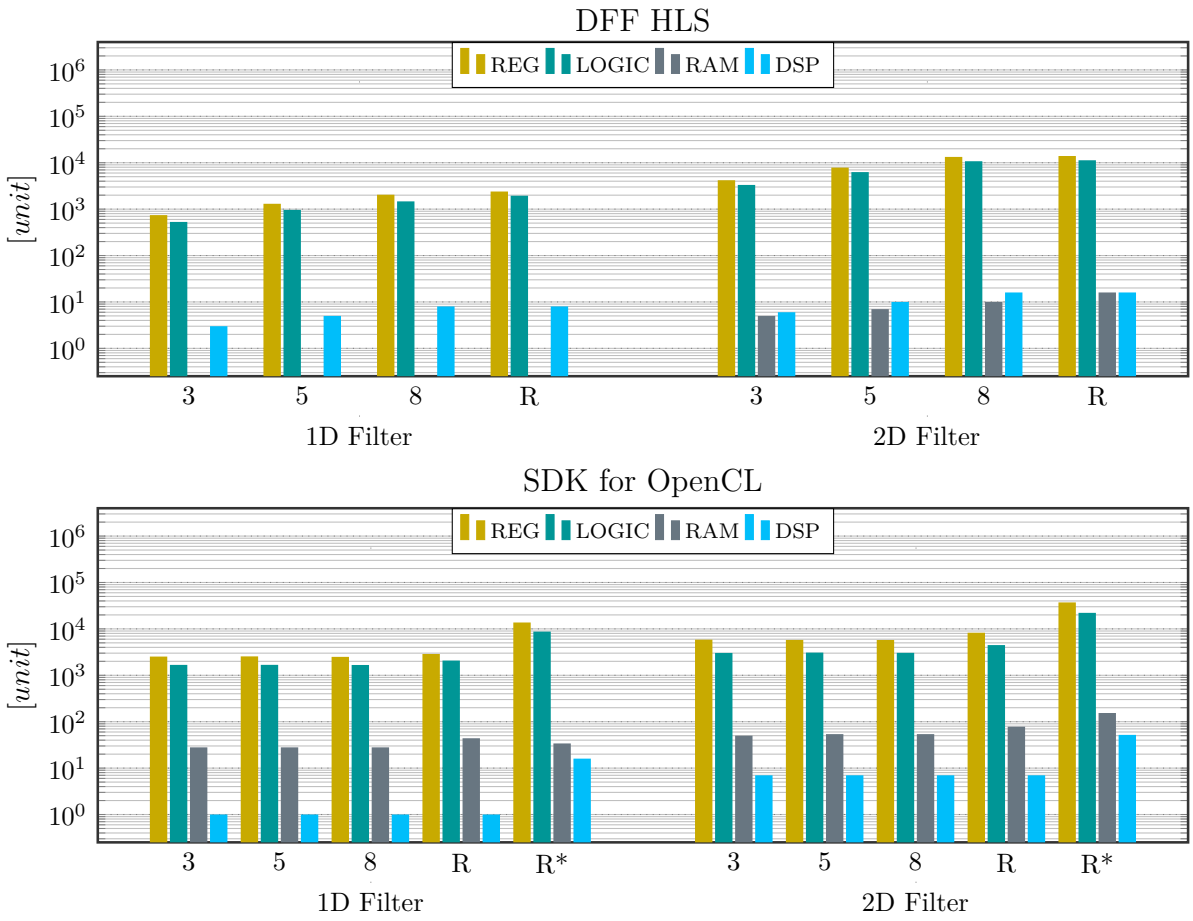


Figure 4.4 – DFF HLS vs. SDK for OpenCL on Intel target FPGA (5CSEMA5) in terms of Intel resources: REG=register, LOGIC=ALUT, RAM=M10K, DSP=DSP. Data are shown on a logarithmic scale with base of 10.

of several metrics (latency, throughput and memory utilization) evaluated with respect to changes in software parameters. On the contrary, MDC provides a model-to-model compiler capable of merging several dataflow applications, as well as a dataflow-to-hardware synthesizer that implements CGR systems. Moreover, MDC profiles CGR system configurations, providing different metrics (area, power, frequency) and includes a power manager that offers clock- and power-gating techniques for improving energy efficiency. In Section 4.2.1, the proposed tool integration aimed at exploiting the features of SPIDER and MDC is proposed in order to respectively manage SW and HW reconfiguration at runtime. Tool integration is also based on the use of DF models with similar properties in order to provide response time estimation. The main idea behind this integration is to

use CGR DSAs as slave processing elements in the target system, and re-schedule these PEs from a host SW-programmed processor at runtime using models of the instantaneous HW behavior in terms of execution time. The SW application has been designed using a PiSDF specification and the toolchain composed of the design-time tool PREESM and SPIDER (integrating the automatic code generation of PAPIFY [1] processor event monitoring [Mad+18]). In this context, actors exchange tokens through FIFOs depending on the feasible working points of the application scenario. The configuration parameters set at design-time and run-time are respectively called *static* and *dynamic*. The latter ones imply on-the-fly re-scheduling and re-mapping when their values change.

In Sections 4.2.2 and 4.2.3 are presented two PoCs implemented for testing the tool integration on the Image Processing and Inverse Kinematics (IK) applications. In both cases, the system is based on the Zynq-7000 XC7Z020CLG484 device running Linux. This consists of SW and HW parts, both modeled as dataflow networks. The SW Application has been mapped upon the two ARM Cortex A9 cores available on the target board. The HW accelerator has been implemented on the Programmable Logic of the FPGA as a CGR DSA.

4.2.1 CGR Adaptation Framework

The proposed framework has been composed from a combination of the state-of-the-art functionalities within the H2020 CERBERO project. Indeed, a main objective of the CERBERO project is to demonstrate an extended adaptation of the CPS calculation to the system state as well as to its environment, adaptation provided by an autonomous reconfiguration engine. The adaptation architecture can be illustrated as in Figure 4.5. An application graph, conforming to a PiSDF MoC, is dynamically scheduled by SPIDER. Depending on the scheduling, a hardware system composed of ARM cores and CGR accelerators (implemented in Xilinx or Intel modern SoC FPGAs) performs the computation. HW and SW monitoring provides feedback to SPIDER about the current execution of the tasks. Regarding the monitoring, this feature has been provided through the integration of PAPIFY, an event-based performance monitoring tool [Mad+18], and MDC [Fan+19b]. In addition, reconfiguration/re-scheduling can also be triggered by sensors in order to adapt the computing layer to the environment changes or system needs.

Figure 4.6 depicts the proposed design flow more in details. In particular, once obtained

1. `https://github.com/Papify`

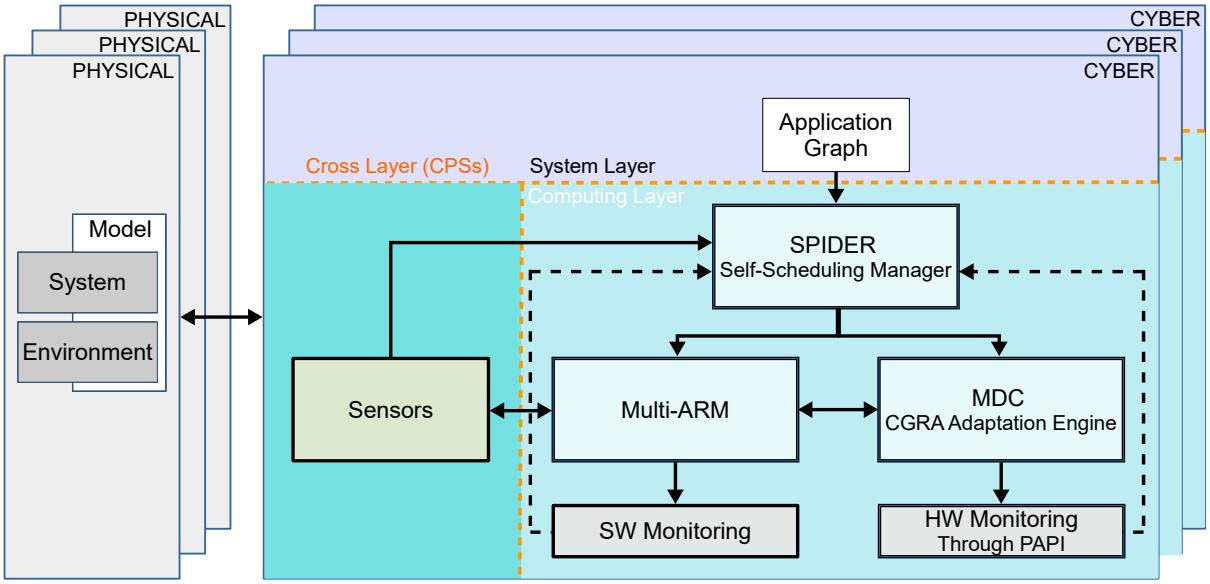


Figure 4.5 – Adaptation scheme as proposed in the H2020 CERBERO European Project.

the srDAG from the PiSDF, the execution of the tasks is delegated to the SW PEs present in the architecture. On the other hand, the HW functionalities of the task to be accelerated ($a6$) are described as DFNs (α, β and γ). With the merging process, these are implemented by the multi-DF network, whose CGRA is generated. The selection of its functionality is performed at the SW level of the accelerated task, which sends it as input to the configuration manager of the CGRA every launch.

4.2.2 Proof of Concept on Image Processing Application with DFF HLS

As a proof of concept, the proposed toolchain is evaluated through an application for Image Processing, involving a multi-functional accelerator for edge detection, able to compute two different filters: *Sobel* and *Roberts*. These are based on discrete first-order differentiation operators, in which the boundary of an object is the difference of the intensity levels in its pixels with respect to the surrounding pixels. These operators are applied to evaluate the gradient image (G), that corresponds to the magnitude of the edge. The computation consists of a convolution of a 3x3 kernel (k) with the source image (A) for *Sobel*, while for *Roberts* k is 2x2: $G = k * A$. This PoC has been used in order to

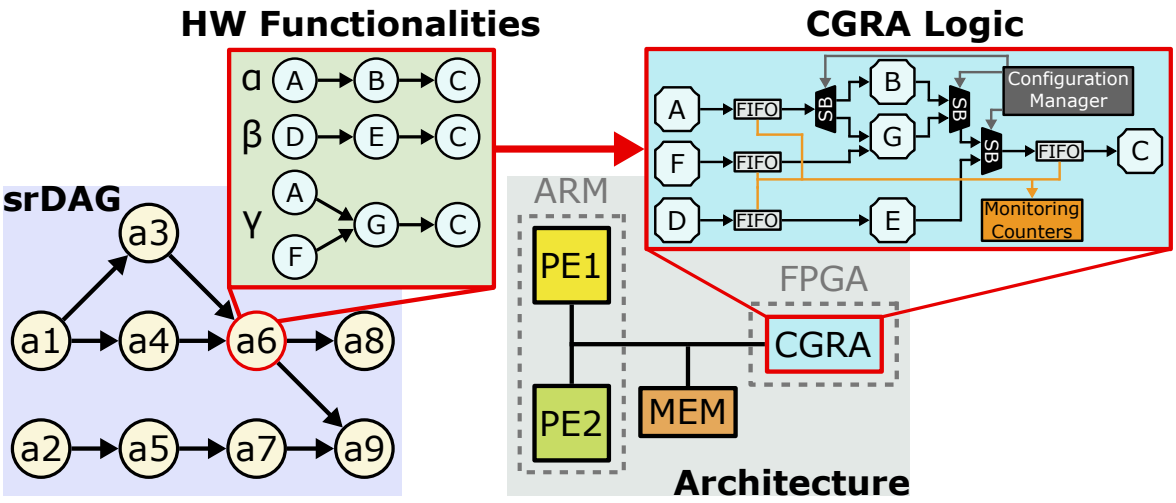


Figure 4.6 – Overview of the heterogeneous HW/SW reconfigurable system.

assess a design flow in which a combination among PAPIFY and MDC has been proposed [Fan+19b].

SW Application

The use-case algorithm depicted in Figure 4.7 can be described as follows:

— Given as an input to the actor *Read_YUV*, a YUV video is read frame by frame, where the number of rows and columns correspond to *height* and *width* parameters respectively. Filtering is applied only to the Y component, while the other ones are directly sent to be displayed.

— Before the edge detection, the block *Split* divides the image in slices depending on the degree of exploitable parallelism. In this assessment, having available one single HW accelerator, no adaptation has been implemented in this sense ($nbSlice = 1$, that is $sliceHeight = height$).

— At this point, verified the on-the-fly selected *kernel* (set by *IdSetter*) among *Sobel* and *Roberts*, an initialization phase is performed in *EdgeMDC_1*. In this phase, the processing data and the communication with the accelerator (through the Direct Memory Access) are handled.

— Then, processing occurs by blocks of pixels of a size suitable for the accelerator specifications (in the assessed example, 32×32). *EdgeMDC_2* sends a number of blocks corresponding to $width\_blk \times height\_blk$ to the *EdgeMDC_hw_filter*, which forwards

the data to the coprocessor. Therefore, *EdgeMDC_3* receives the result of each iteration, which is collected in *EdgeMDC_4*.

— Finally, the filtered frame is merged and displayed with the applied type of kernel and the execution time expressed in Frames per Second (FpS).

With respect to the mapping strategy, SPIDER handles all SW tasks taking into account the constraints given as input by the application designer. In the evaluated case, the actors performing splitting and merging have to be executed onto the same core. Moreover, SPIDER has managed 305 instances of the single-rate graph. Indeed, 8 actors are executed 1 time per firing, and 99 times the other 3 ones (*EdgeMDC_2*, *EdgeMDC_hw_filter*, and *EdgeMDC_3*), since 99 32×32 blocks are present in the frame size considered in this assessment (352x288 pixels). Regarding the actual filtering, this has been accelerated on HW, as explained in the next section.

HW accelerator

As described in Section 2.4.2, MDC takes as input the DF descriptions of the applications to be accelerated. These DF specifications has been processed by MDC, to generate a CGR DSA able to compute both *Sobel* and *Roberts* algorithms, which has been automatically embedded into the ready-to-use Xilinx IP. The CGR accelerator has been implemented considering the design flow proposed in Section 4.1.1.

4.2.3 Proof of Concept on an Inverse Kinematics Application with MDC

An additional PoC has been implemented by using the CGR Adaptation Framework for an IK algorithm. The application considers the Dumped-Least Squares (DLS) algorithm in order to control a robotic arm [Fan+19a]. In order to demonstrate the capabilities of MDC, a previous version of this PoC have been used as a baseline in the assessment proposed in [Sau+20b].

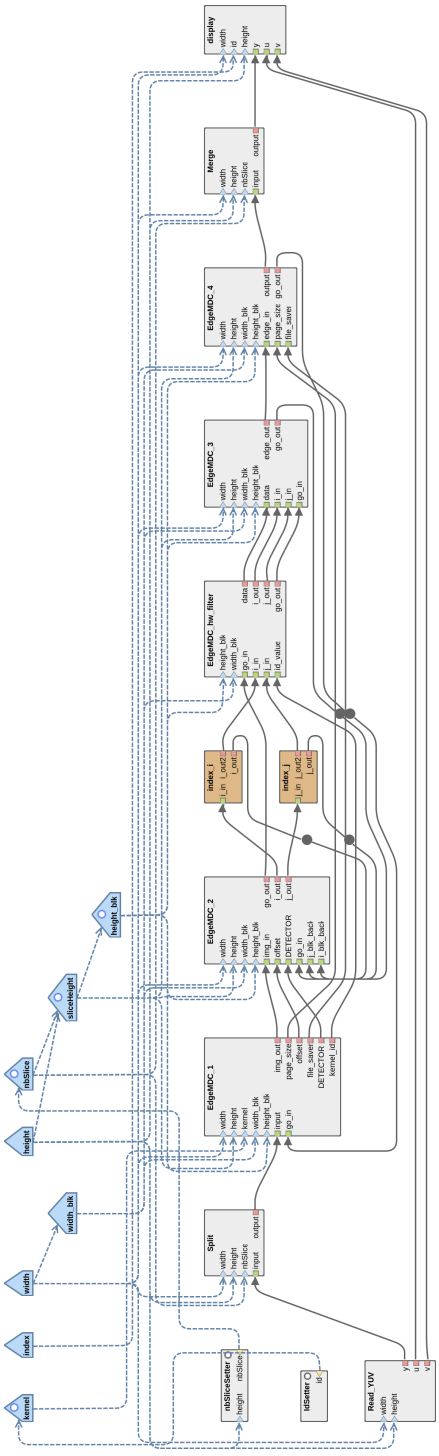


Figure 4.7 – DF description of the SW application, modeled using PREESM. Actors and routing blocks are respectively represented by gray and orange boxes. Blue pentagons correspond to the PiSDF parameters that, given as inputs to the actors, establish their specific functionalities. In particular, parameters with white circle vary at runtime.Connections among blocks depict the data token transfer links (gray wires) and the dependencies from the parameters (blue dashed lines).

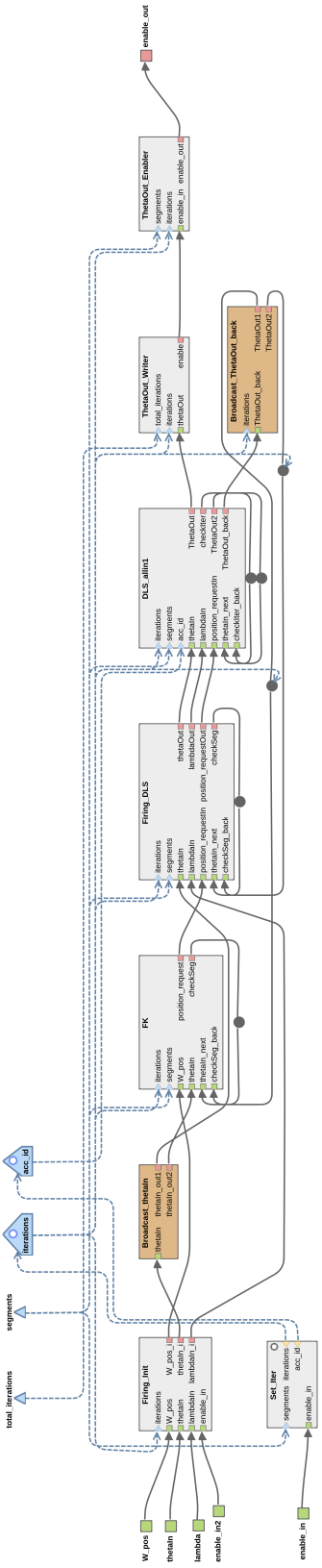


Figure 4.8 – PiSDF description of the subgraph *DLS_Loop*.

SW Application

Figure 4.9 and Figure 4.8 show the DF description of the DLS application and of its unique subgraph respectively, that have been designed by using PREESM and SPIDER. The evaluated implementation can be explained as follows:

— A configuration actor (*DLS_PreProcessing*) reads the inputs given by the user related to the accuracy and the required points of the robotic arm trajectory. From the distance between subsequent input points, it calculates the number of iterations required to the DLS to achieve every specific point, given input accuracy. Then, it writes a file ("segmentsInfo.txt") in which all the points are reported with the related number of iterations. Moreover, it evaluates the total number of iterations needed to perform the entire movement (sum of the iterations of each point). However, its main work is to set the dynamic parameters: number of input segments/points (segments) and number of total iterations (*total_iterations*).

— The actor *DLS_init* fires one time per graph firing, only to read and send all the point information contained in the "segmentsInfo.txt", and enable the N firings of the next actor *Firing_Init* (where N corresponds to the parameter segments).

— For each point of the trajectory, *Set_Iter* updates the number of iterations (*iterations*), on which the fifo sizes and delays (grey circles on the edges) depend. Moreover, depending on the battery level, it triggers high or low performance of the accelerator by setting its ID (parameter *acc_id*).

— *FK* provides the proper angles of the joints associated to the starting point of the trajectory (that is statically defined).

— The actor *Firing_DLS* handles input data tokens to send to the *DLS_allin1*.

— For the whole trajectory, *DLS_allin1* fires *iterations* times for each input point, performing the DLS algorithm and sending the output to the *ThetaOut_Writer* that stores it into the file “thetas.txt” for each DLS iteration.

— *ThetaOut_Enabler* fires at the end of every segment present in the trajectory only to enable the *DataSender*. This actor properly sets the angles of the joints, send the data of the whole trajectory to the arm, and enables a new reading of the input in the configurator (*DLS_PreProcessing*).

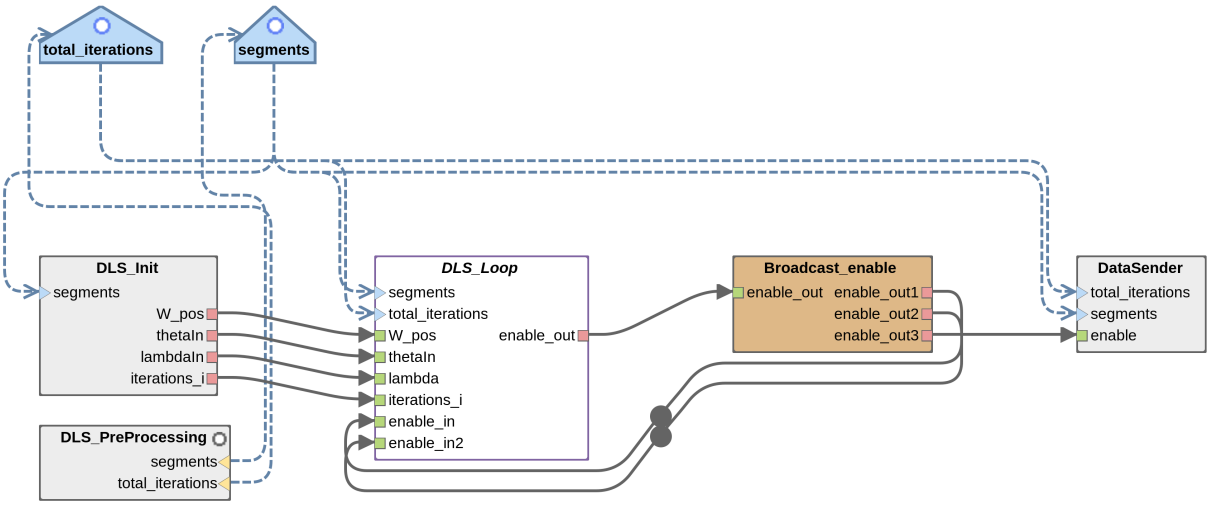


Figure 4.9 – PiSDF description of the DLS application for run-time configuration using SPIDER.

HW Accelerator

The CGR DSA corresponds to the hardware counterpart of the actor *DLS_allin1* within the *DLS_Loop*. This can be reconfigured in two different working modes in term of execution time: baseline and high performance. In this case, the HDL specifications of the actors present in the CGR DSA (generated by MDC) have been derived from Vivado HLS, and not by using the design flow proposed in Section 4.1.1.

4.2.4 Section Remarks

In a signal processing HW context, efficiency and flexibility in terms of functionalities can be reached by considering CGR DSAs because these accelerators combine a reconfigurable datapath and local data exchanges without reaching a main memory. The DFF HLS has proven capable of providing these flexible devices for both Xilinx and Intel FPGAs and predicting their latency at IP level. In order to exploit CGR acceleration at system level, this design flow has been integrated with the proposed flow in a HW/SW context. Indeed, the complementary properties of MDC and SPIDER have enabled a novel approach based on the dataflow modularity and the runtime handling of the HW reconfiguration. In order to support fast reconfiguration and the correct execution of the application, SW tasks and CGR acceleration can be combined by using the proposed flow that extends the features provided by the state-of-the-art SPIDER and MDC tools. In

addition, this integration has supported the creation of the presented PoCs that have been used in the assessment of further works within the H2020 CERBERO project. In current version, the complexity due to the execution of parallel applications on heterogeneous architectures leads to a loss of predictability at system level. Moreover, the dynamic management of parametric applications requires a specific strategy to perform adaptation. For these reasons, a method aimed at evaluating the execution time of the whole DAG to be implemented in both HW and SW is needed, as mentioned in Section 3.2 and proposed in Chapter 5.

4.3 Chapter Remarks

This chapter contains two main parts respectively aimed at i) improving response time predictability in the design of CGR IPs (Section 4.1), and ii) providing a framework in which these accelerators can be integrated in higher-level DF-based parametric applications involving hardware and software tasks (Section 4.2). From the HW perspective, reconfiguration can be applied with respect to the response times of each operating mode, since this metric is known and understandable during the design phases (even before the synthesis process associated with a specific target). Nevertheless, the execution of a functionality with multiple SW tasks on a general-purpose architecture implies a context difficult to determine. Therefore, in order to offer a support to the proposed adaptation framework, a strategy based on the MoA approach have been identified (as described in Chapter 6). The proposed framework consists of a combination of MDC and SPIDER which is not completely automatic, and several improvements can be developed. As an example, an implementation of different type of dynamic parameters associated with the reconfiguration of the accelerators is required, since these parameters currently imply re-scheduling and re-mapping at each change of their values.

Chapter 5

Contribution 2: PathFinder: studying the application latency-causing activity

The PathFinder method introduced in Section 3.2 computes the application activity determining response time from applications modeled by such DAGs of tasks, tagged with timings for both task execution and message passing. In experimental results, these DAGs of tasks are generated from DF modeled applications using the PiSDF MoC. The chosen definition of response time corresponds to the value W in the queuing formula [Lit61]. The value W gives the *expected time spent by a [data] unit in the system* as it passes from a data source to a data sink. In order to make the study more practical, several data sources and sinks can be considered, determined by the system designer and based on the objectives of the system. The derived response time is the maximum of these independent latencies. As an example, a near-sensor stereo matching algorithm that computes a depth map from two views of a scene takes as data sources two images from sensors and computes a unique map representing the depth of the scene at each pixel. PathFinder considers system execution latency as the time difference $t_p - min(t_{a1}, t_{a2})$, where t_{a1} and t_{a2} represent the arrivals of the first pixel from the two cameras, and t_p corresponds to the production of the last pixel of the depth map.

To the extent of our knowledge, this work constitutes the first effort to decompose the determining factors of system execution latency in an MPSoC. The main contributions of this chapter are:

— a model of the application activity causing response time in the form of an LLP;

— a method to extract an LLP from an application task DAG mapped upon an MPSoC with a post-execution analysis;

— an assessment of the model and method on large, functional task graphs.

5.1 Design Flow

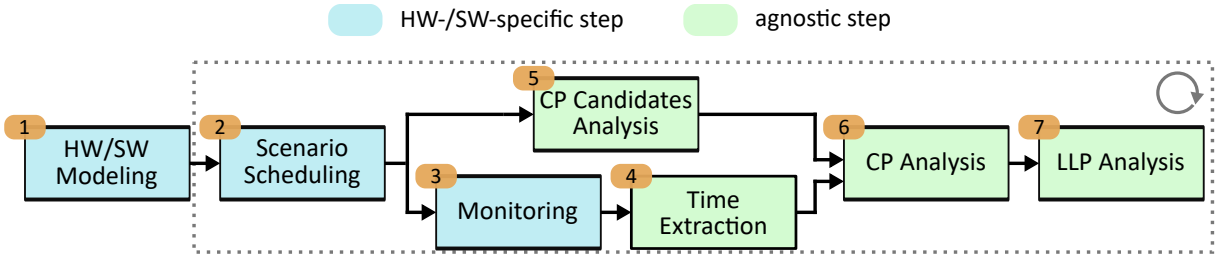


Figure 5.1 – Design flow implementing the PathFinder method.

In this section, the design flow for estimating LLP latency for a parametric application described as a task graph is proposed. As depicted in Figure 5.1, this starts from the creation of the application and the modeling of the target architecture (1). Next phases can be performed in loop for each scenario, starting from the scheduling (2). The application execution is monitored (3). A user that wants to apply the proposed flow needs to manage these 3 HW- and SW-specific steps by utilizing tools in order to schedule, map and monitor the DF-based application onto a heterogeneous system. After that, a fully-automated chain of scripts analyzing system execution latency can be launched. At first, this process provides a list of CP candidates depending on the DAG analysis, and then adding interferences to a chosen CP in order to build the LLP that approximates the response time of the functionality. Indeed, while CP is the list of tasks causing response time in an idealized architecture with an unlimited number of PE, LLP is the list of tasks and communications determining response time in a real MPSoC, including many interferences of different kinds (as discussed in Section 3.2.2). The input list provided by the user for these phases consist of:

— the architectural information (number of PEs, and types, corresponding to the names, of PEs);

— the srDAG (with parameters indicating how to explore it);

— the running and monitoring information (number of executions and measures);

— the selected statistical metrics (e.g. average) on which to base the timing characterization and the LLP analysis.

Thus, depending on the chosen metric, the timing cost of the actor and edge instances are extracted from measurements (4). In parallel, the CP Candidates analysis explained in Section 5.1.5 can be evaluated (5). Following to this method, the CP is obtained starting

from the monitored actor times for a desired execution (6). Finally, the LLP analysis is performed (7). In the rest of this section, flow steps are described in details.

5.1.1 HW/SW Modeling for LLP Analysis

In order to feed the LLP analysis, the model of the HW system and the application have to be provided to the scheduler. The architectural information consists of the specified number of PEs of different types (already known at the level C described in Section 3.2.2). On the other hand, the application is modeled as a srDAG (as in Section 3.2.1).

5.1.2 Scheduling Scenarios

In contexts in which instantaneous system requirements imply the reconfiguration of the application, functional changes can lead to application graphs with different numbers of actors and paths. For this reason, it can be useful for the user to identify specific operating modes (scenarios) of the application. The LLP analysis can then be focused separately on the srDAG structures associated with each individual scenario. This favors the awareness of the user about the impact of multiple instances in the execution in terms of speedup and interferences. After a task characterization on multiple PE types from single-core executions, the proposed flow relies on the scheduling and mapping algorithm of the compilation tool (in our case PREESM) in order to find unconstrained solutions, only dependent on the number of PEs, and on time characterization of the actors and the communication cost. Moreover, in order to provide the most flexibility in executing actors, the HW configuration corresponds to the exploitation of all the PEs present in the HW target. Nevertheless, this corresponds to the worst-case configuration, since this choice could imply a delay in the execution of the actors and their data exchanges.

5.1.3 Measuring Local Timings

Due to the timing variability of the srDAG instances, multiple measurements are required for their post-characterization. Indeed, starting and ending times of the tasks are obtained by a system monitoring, and used to obtain a CP linked to the particular running process. This CP takes part in the LLP evaluation, and computed as proposed in Section 5.1.6. Moreover, the measures feed the statistical latency evaluation of the srDAG.

5.1.4 Latency Post-Characterization and Relevant Execution

Depending on the objective of the exploration, the choice of two statistical metrics of interest leads respectively to: i) the post-characterization of the time contributions for DAG actors and edges, and ii) the analysis based on a specific monitored execution that more than the others represents the chosen metric. This pair of metrics customizes the latency evaluation considered in the presented flow.

5.1.5 Finding the Critical Path Candidates

Starting from the application srDAG, several ways can be considered to obtain CP candidates. In this work, complete and selective analyses have been evaluated (see Tables 5.1 and 5.2). The complete analyses does not require a-priori knowledge of the application and provides the user with all the paths present in the graph. However, this could not always be convenient since scalability issues may occur. To avoid these, in the selective analysis, the evaluation is limited by setting the maximum number of paths per source to be considered. Moreover, being familiar with the application favors the reduction of computational complexity by selecting sources and sinks of the paths that have to be analyzed.

Description of the Algorithms

Starting from each graph source actor (i.e. without predecessor), the search algorithm implies to find the next nodes connected to it (lines 14 and 16 in algorithms of Tables 5.1 and 5.2 respectively). This continues until all paths for each sink actor (i.e. without successor) has been found (see procedure f_nxt, lines from 1 to 10 in algorithm of Table 5.1). The heuristic version of such procedure evaluates only N paths for each selected input (see f_nxt_N, lines from 1 to 12 in algorithm of Table 5.2). Procedures sel_graph and sel_edges (lines 13 and 14) are used to select only the part of the srDAG affected by the paths with the chosen sources (Sel_Sources) and sinks (Sel_Sinks) with their associated edges (Sel_Edges), instead of considering all the sources (All_Sources), sinks (All_Sinks) and edges (All_Edges).

Complexity of the Algorithms

A full exploration of the srDAG can easily lead to an explosion of the investigation timing depending on the number of nodes and links among them. Indeed, the algorithm

In	srDAG
Out	all CP Candidates
Pros	no application knowledge is needed
Cons	poor scalability

Algo

```
1  Procedure f_nxt(starting_node,Links,Paths)
2      for i ← 1 to size(Links) do
3          next_node = get_target(starting_node,Links[i]);
4          if next_node then
5              update_paths(Paths);
6              Paths = f_nxt(next_node,Links,Paths);
7          end
8      end
9      return Paths;
10 end
11 All_Sources = find_all_sources(srDAG);
12 All_Edges = find_all_edges(srDAG);
13 for i ← 1 to size(All_Sources) do
14     CP_Candidates = f_nxt(All_Sources[i],All_Edges,CP_Candidates);
15 end
16 return CP_Candidates;
```

Table 5.1 – Overview of the Complete CP Candidates Analysis.

of Table 5.1 presents a complexity of $\mathcal{O}(s \cdot e \cdot n)$, where s = All_Sources, e = All_Edges, and n is the maximum length of the paths in the srDAG. Hence, to make the analysis be tractable working on large graphs, the search space should be strategically pruned. The analysis can be limited to $\mathcal{O}(s' \cdot e' \cdot n')$, with $s' \leq s$, $e' \leq e$, $n' \leq n$, $s' =$ Sel_Sources, $e' =$ Sel_Edges, and n' the maximum length of the paths in the limited srDAG. Moreover, the space complexity can be reduced to $\mathcal{O}(s' \cdot e'' \cdot n'')$ (with $e'' \leq e'$, $n'' \leq n'$) by limiting the searching to only N CP candidates per source. In the current version of PathFinder, the parameters s' and n'' that limit the complexity of the analysis are given as inputs by the user, while e' and e'') are obtained in dependence on these.

5.1.6 Choosing the Most Probable CP

Among the CP candidates collected in the previous phase, the critical (longest) one can be detected with strategies based on the available information. In the proposed method (see Table 5.3), details of the monitored execution are exploited: scheduling, mapping, starting and ending times of each actor (available at level E). CP provides an estimation

In	srDAG, selected sources/sinks
Out	selected CP Candidates
Pros	tractable complexity
Cons	application knowledge is needed

Algo

```
1  Procedure f_nxt_N(starting_node,Links,Paths,N)
2      if size(Paths) < N then
3          for i ← 1 to size(Links) do
4              next_node = get_target(starting_node,Links[i]);
5              if next_node then
6                  update_paths(Paths);
7                  Paths = f_nxt_N(next_node,Links,Paths,N);
8              end
9          end
10     end
11     return Paths;
12 end
13 Sel_srDAG = sel_graph(srDAG,Sel_Sources,Sel_Sinks);
14 Sel_Edges = sel_edges(Sel_srDAG);
15 for i ← 1 to size(Sel_Sources) do
16     CP_Candidates_i = f_nxt_N(Sel_Sources[i],Sel_Edges,CP_Candidates,N);
17     CP_Candidates = append_paths(CP_Candidates,CP_Candidates_i);
18 end
19 return CP_Candidates;
```

Table 5.2 – Overview of the Selective CP Candidates Analysis.

of the response time with a poor accuracy, that needs to be integrated in an LLP by adding interferences (as in Section 5.1.7). However, this method makes it possible to examine latency variability of the CP tasks during the execution. Among the paths provided as input (CP_Candidates), algorithm of Table 5.3 detects the CP (CP) as that path with the maximum difference (max_diff) between the end time of its last actor and the start time of its first one (by the procedure get_diff). As a next step, the Gantt chart describing scheduling/mapping of the real execution (Mon_Gantt) is explored. In addition, the processing ratio (path_proc_ratio) between the latency due to the all of the actors in the path (path_lat) and the one due to the whole path execution (end_start_diff) determines the choice of the CP. If more paths with the same end-to-start difference are present, the choice falls on that one with the largest processing ratio. The logic behind this decision aims to reduce the impact of the communication links (not considered in this version of the PathFinder) that can occur in the time windows present among executions of connected CP actors. Communication times have been ignored because all experiments have

been conducted on an MPSoC with shared memory for which inter-PE communications use loads/stores which latencies are measured within actor execution time.

In	CP Candidates, measured execution time tags
Out	CP, CP latency
Pros	analysis of runtime latency variability
Cons	poor accuracy, running information needed

Algo

```
1  max_exec_time = 0;
2  for i ← 1 to size(CP_Candidates) do
3  |  for j ← 1 to legth(CP_Candidates[i]) do
4  |  |  actor_lat = Mon_Actor_Lats[CP_Candidates[i, j]];
5  |  |  path_lat = path_lat + actor_lat;
6  |  end
7  |  end_start_diff = get_diff(Mon_Gantt,CP_Candidates[i]);
8  |  if i == 1 then
9  |  |  path_proc_ratio = path_lat/end_start_diff;
10 |  else
11 |  |  path_proc_ratio = path_lat/max_diff;
12 |  end
13 |  if end_start_diff > max_diff then
14 |  |  max_diff = end_start_diff;
15 |  end
16 |  if path_proc_ratio > CP_proc_ratio then
17 |  |  CP_lat = path_lat;
18 |  |  CP_proc_ratio = path_proc_ratio;
19 |  |  CP = CP_Candidates[i];
20 |  end
21 end
22 return CP and CP_lat;
```

Table 5.3 – Overview of the A-Posteriori CP Analysis.

5.1.7 Finding the LLP

Adding the interference to the CP analysis leads to a better awareness about the determining factors response time, exploiting the information of the level E presented in Section 3.2.2. Nevertheless, this further investigation increases the evaluation time and the complexity of the analysis. Indeed, for each element of the CP, an identification of potential interference is required (see Table 5.4). The different types of interferences have specific weights, and one of them can dominate over the others. Thus, a selective evaluation (get_sel_int in algorithm of Table 5.4) can satisfactorily approximate a more

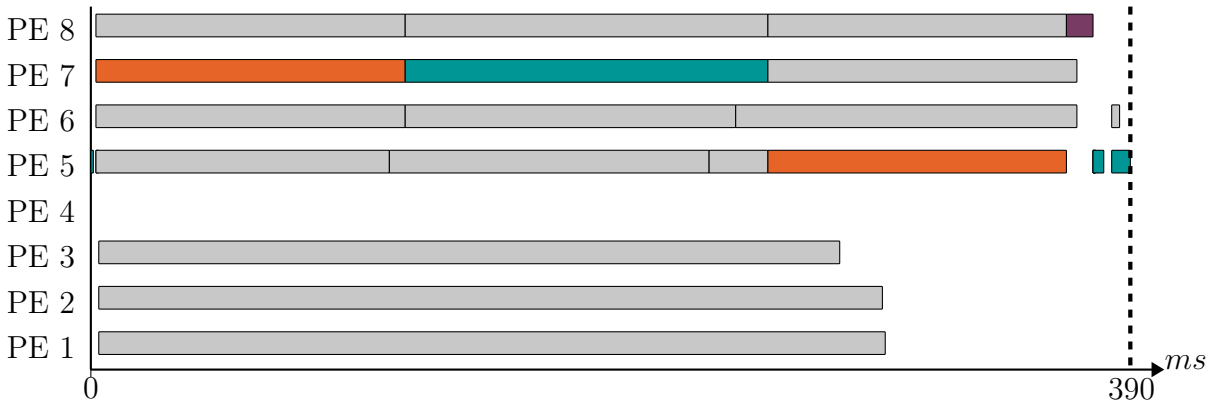
complete one, at the same time reducing the time and complexity of the analysis. In this work, interferences due to scheduling and dependency (defined in the Section 3.2) have been added to the selective LLP analysis. In the case of overlapping, only that one due to the scheduling is counted. Moreover, the dependency interference is computed in a recursive manner and, in case of overlapping, considering only one contribution at a time. Figure 5.2 shows how LLP appears in three Gantt charts representing the real execution of one DAG iteration of three different applications mapped on a heterogeneous 8-PEs MPSoC, in which PEs with index from 1 to 4 are less performing in terms of ISA than PEs with index from 5 to 8. LLP is represented as a succession of the execution time contributions of tasks associated with the CP (in teal color), scheduling interference (in orange) and dependency interference (in purple). The considered mappings show different characteristics in terms of LLP. Indeed, Figure 5.2A shows a work-dominated application, that is, where the CP contribution in the LLP is lower than the interferences. In Figure 5.2B, the LLP is the result of a balanced mix of both contributions due to CP and interferences. Finally, Figure 5.2C depicts an LLP dominated by the CP (or span).

In	CP, CP latency, execution times of actors, Gantt Chart
Out	LLP, LLP latency
Pros	medium complexity
Cons	high accuracy

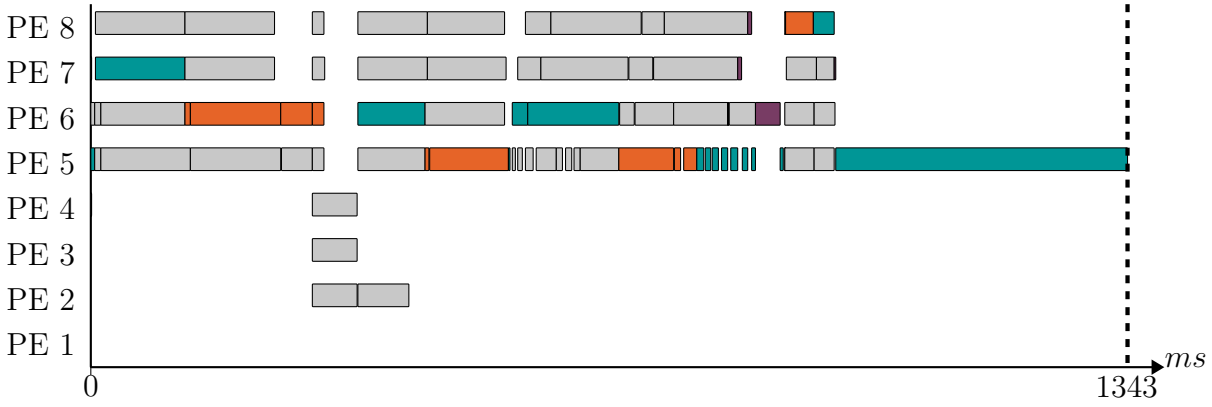
Algo

```
1  k = 1;
2  for i ← 1 to legth(CP) do
3  |   LLP[k] = CP[i];
4  |   k = k + 1;
5  |   for j ← 1 to size(Actor_Lats) do
6  |   |   actor_int = get_sel_int(CP[i],Actor_Lats[j],Gantt);
7  |   |   if actor_int > 0 then
8  |   |   |   CP_int = CP_int + actor_int;
9  |   |   |   LLP[k] = Actor_Lats[j];
10 |   |   |   k = k + 1;
11 |   |   end
12 |   end
13 end
14 LLP_lat = CP_lat + CP_int;
15 return LLP and LLP_lat;
```

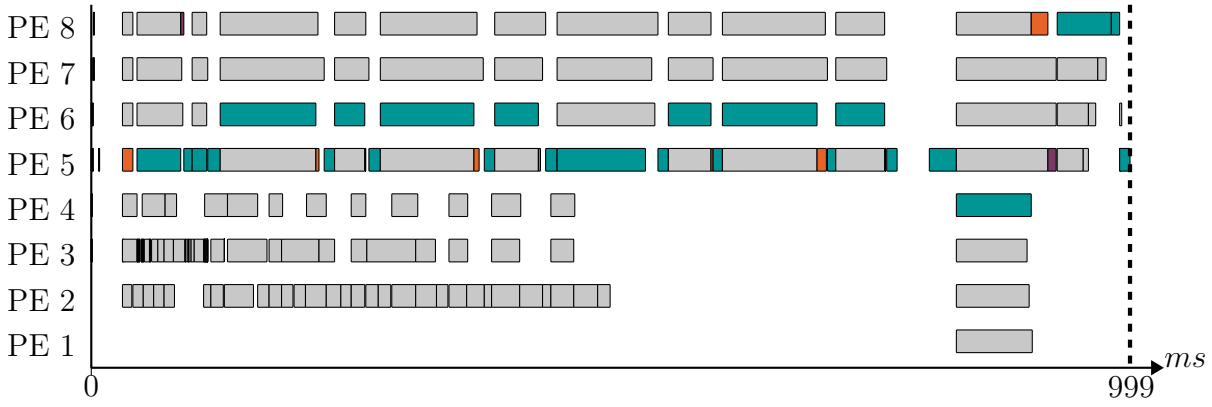
Table 5.4 – Overview of the Selective LLP Analysis.



(A) Work-Dominated Application



(B) Balanced Work and Span Application



(C) Span-Dominated Application

Figure 5.2 – Example of one iteration of DAG execution, represented as Gantt charts, of work-dominated, balanced work and span, span-dominated applications mapped on an heterogeneous MPSoC. These proposed execution views are associated with three DAGs respectively composed of 22, 74 and 172 actors (as will be assessed in Section 5.2). LLP tasks are highlighted in teal, orange and purple, depending on the type of contribution, as tasks associated with: CP, scheduling interference, and dependency interference respectively.

5.2 Assessment

In this section, a comparison among the approaches described in Section 3.2 and the LLP analysis is proposed. The flow described in Section 5.1 has been assessed with 3 different PiSDF applications evaluated in several scenarios and mapped onto an ODROID-XU3 board [Har]. As explained before, PiSDF [Des+13] is a DF MoC that models synchronization among tasks and algorithmic semantics handled by using high-level parameters. By using this MoC, the user can vary the consumption/production rates to obtain different structures of the srDAG. Indeed, each configuration of the PiSDF parameters represents a specific scenario of the application, having a certain number of actor instances. The PiSDF graphs of the chosen applications have been designed in PREESM [Pel+14], which embeds a non-preemptive ASAP multi-core list scheduling targeting latency minimization [KA99]. The monitoring consists of 101 executions with the highest priority (available for a process in the Linux environment) for each evaluated scenario. Among these, the execution that best fits the mean value of the 100 measures (excluding the 1st execution to avoid typical initial events, such as first input load into the RAM) is selected to represent the response time of the scenario. Moreover, the same metric has been used for the characterization of the srDAG instances. About CP candidates analysis, only 50 random paths per graph source have been explored, we show that represent a good trade-off between complexity and DAG exploration for the use-case applications.

Regarding the target platform, the experimental setup processor is a Samsung Exynos 5422 composed of 8 ARM cores in a big.LITTLE configuration: 4 cores are of type Cortex-A7 with a cluster frequency up to 1.4GHz, and other 4 cores of type Cortex-A15 with a cluster frequency up to 2GHz. In the experiments, both clocks have been set at their maximum. In order to observe the most difficult scheduling conditions, all the 8 PEs present in the HW target have been used for mapping of the tasks. Indeed, this choice leads to an increased execution time associated with the data exchanges among actors.

5.2.1 Use-case Applications

This study exploits realistic use-case applications. In the context of image/video processing, the selected applications show distinct features in terms of actors, firing, and parallelism. These have been evaluated in 6 different scenarios depending on their parameters in order to change the dataflow graph structure: CP candidates, pipeline actor stages, data parallel tasks.

Example of a Work-Dominated Application — Video Stabilization

This application is *work-dominated* for the considered architecture parallelism. This property means that the response time is influenced more by the number of PEs and architecture properties rather than by application limitations. Video stabilization reduces the effects of undesired fast camera movements due to a shaking camera during video recording. Post-processing techniques can analyse image motion, leading to the generation of a new video in which shaky movements are removed. The author thanks Karol Desnos for providing the PiSDF version of the Video Stabilization application. For each frame, the proposed solution [1] embeds the following steps:

— a frame is read from a YUV stream (*ReadYUV* in Table 5.5);

— the input frame is divided into blocks of BLOCK_HEIGHT by BLOCK_WIDTH pixels (*DivideBlocks*);

— for each block, the minimum squared error is evaluated with respect to its possible position in the previous frame, after a restricted research (limited by MAX_-DELTA_X/Y) that leads to the creation of a motion vector (*ComputeBlockMotion-Vector*, the only one data parallel actor);

— the different motion vectors are analysed in order to identify the dominating motion by using multivariate Gaussian criteria (*FindDominatingMotion*);

— the dominating motion vector is accumulated (*AccumulateMotion*);

— the motion-compensated frame is rendered with two times BORDER additional pixels with respect to the input, in order to move within a larger area before leaving the frame (*renderFrame*);

— a hash value is assigned to the rendered frame (*MD5*);

— the rendered frame is stored in order to create the post-processed video (*WriteYUV*).

Table 5.5 shows the repetitions of the 8 actors present in the PiSDF, application model corresponding to the number of actor instances in the srDAG. Each column is associated to a chosen scenario that depends on the values of the parameters BLOCK_-HEIGHT, BLOCK_WIDTH and BORDER mentioned before. In this context, since the task graph presents only one data parallel actor (*ComputeBlockMotionVector*), the split of its associated work leads to the generation of a certain interference that implies a decoupling between the CP and the real latency.

1. `github.com/preesm/preesm-apps/tree/master/org.ietr.preesm.stabilization`

n	PiSDF Actor	Repetitions					
		256,10	192,200	160,200	128,200	64,200	48,200
1	ReadYUV (src)	1	1	1	1	1	1
2	SG1/SG2/DivideBlocks	1	1	1	1	1	1
3	SG1/SG2/ComputeBlockMotionVector	2	6	8	15	66	120
4	SG1/FindDominatingMotion	1	1	1	1	1	1
5	SG1/AccumulateMotion	1	1	1	1	1	1
6	SG1/renderFrame	1	1	1	1	1	1
7	MD5 (snk)	1	1	1	1	1	1
8	WriteYUV (snk)	1	1	1	1	1	1
	No. of actor instances	9	13	15	22	73	127
	No. of edge instances	41	57	65	93	297	513

Table 5.5 – Actor repetitions in one DAG graph iteration in the chosen scenarios for the Video Stabilization application. Each scenarios is represented by 2 parameters, with the following string: p1,p2; where, p1 is the block size (p1xp1, where BLOCK_HEIGHT == BLOCK_WIDTH), p2 is the BORDER parameter. For this application, actors within a subgraph (SG) are preceded by SG1 and/or SG2 (respectively named Stabilization and ComputeBlockMotionVectorss in the PiSDF). When the actor name is specified by (src) or (snk), this is a source actor or a sink actor of the graph respectively.

Example of a Balanced Work and Span Application — Stereo Matching

For the considered platform, the Stereo Matching application has complex behaviors in terms of response time: latency is influenced by both application CP and architecture number of PEs. From the comparison of two images from two different poses of the same scene, the Stereo Matching application obtains the scene depth information in the form of a disparity map. Indeed, disparity map pixels represent the distance between the locations of the same pixel in the two regularized views. The considered implementation [2] can be described as follows:

— two RGB images are read as PPM files (*Read_PPM0* and *Read_PPM1* in Table 5.6);

— a grayscale conversion is applied to both, left and right views (*RGB2Gray_L* and *RGB2Gray_R*);

— every pixel of the images is compared to its 8 neighbors in order to obtain an 8-bit signature, where the bit is set to 1 if the value of the neighbor is greater, otherwise to 0 (*Census_L* and *Census_R*);

2. `github.com/preesm/preesm-apps/tree/master/stereo/org.ietr.preesm.stereo`

— an array of offsets used in the computation of the depth map is generated (*OffsetGen*);

— horizontal and vertical weights are produced for each pixel, depending on the neighboring pixels and the offsets, and firing NbIterations times (*ComputeHorWeights* and *ComputeVertWeights*);

— the disparity level (NbDisparity) is evaluated (*DisparityGen*);

— the grayscale images and their census signatures are combined in order to produce the cost of matching of a specific pixel of the left image with its corresponding pixel in the right image shifted by the disparity level (*CostConstruction*);

— for each pixel, an aggregation of the the horizontal and vertical disparity error for several offsets is performed (*AggregateCost*);

— a disparity map is generated by computing the disparity of the input cost map from the lowest matching cost for each pixel (*disparitySelect*);

— the disparity map is divided depending on the parameter NbSlice (*Split*);

— a 3-by-3 pixels median filter fires NbSlice times in order to smooth the result (*Median_Filter*);

— the result image is written in the PPM format (*Write_PPM*).

For each selected scenario, the number of actor firings per DAG iteration are reported in Table 5.6. The application has been evaluated depending on the values of 3 different parameters: NbDisparity, NbIterations and NbSlice. Besides the mentioned firing dependencies (of the actors *ComputeHorWeights*, *ComputeVertWeights*, and *Median_Filter*), the NbDisparity affects the pipelined repetitions of the task chain composed by *CostConstruction*, *AggregateCost* and *disparitySelect*. The task graph presents mixed characteristics between the exploitation of its intrinsic parallelism and the tuning of its work to reach a better depth map quality. These characteristics lead to a proportional relationship between the CP and the real latency. Nevertheless, since the amount of interference is not negligible, the accuracy of the CP as a model of response time decreases with the number of actor instances present in the srDAG.

Example of a Span-Dominated Application — SIFT Point Computation

This application is *span-dominated* as its response time in the architecture considered is strongly characterized by its CP. In the context of computer vision, the Scale Invariant

n	PiSDF Actor	Repetitions					
		2,2,4	2,4,4	2,4,8	4,4,8	16,4,8	64,4,8
1	Read_PPM0 (src)	1	1	1	1	1	1
2	Read_PPM1 (src)	1	1	1	1	1	1
3	RGB2Gray_L	1	1	1	1	1	1
4	RGB2Gray_R	1	1	1	1	1	1
5	Census_L	1	1	1	1	1	1
6	Census_R	1	1	1	1	1	1
7	SG1/OffsetGen (src*)	1	1	1	1	1	1
8	SG1/ComputeHorWeights	2	4	4	4	4	4
9	SG1/ComputeVertWeights	2	4	4	4	4	4
10	SG1/DisparityGen (src*)	1	1	1	1	1	1
11	SG1/CostConstruction	2	2	2	4	16	64
12	SG1/AggregateCost	2	2	2	4	16	64
13	SG1/disparitySelect	2	2	2	4	16	64
14	Split	1	1	1	1	1	1
15	Median_Filter	4	4	8	8	8	8
16	Write_PPM (snk)	1	1	1	1	1	1
	No. of actor instances	24	28	32	38	74	218
	No. of edge instances	80	94	102	136	331	1099

Table 5.6 – Actor repetitions in one DAG graph iteration in the chosen scenarios for the Stereo Matching application. Each scenario is represented by 3 parameters, with the following string: p1,p2,p3; where, p1 is the disparity level (NbDisparity), p2 is the number of iterations (NbIterations), p3 is the number of slices (NbSlice). For this application, actors within a subgraph (SG) are preceded by SG1 (named Cost_Parallel_Work in the PiSDF). When the actor name is specified by (src) or (snk), this is a source actor or a sink actor of the graph respectively. A * is added to indicate an initialization actor unrelated to the input that can be evaluated as a source node in the I/O path analysis.

Feature Transform (SIFT) is an algorithm used to detect features of an image. Keypoints are extracted by the comparison between corresponding points of the input image and of the same image evaluated in its blurred versions and at different resolutions. The author thanks Alexandre Honorat for providing the PiSDF version of the SIFT Point Computation application. In particular, the evaluated implementation [3] can be explained as follows (see Table 5.7):

— name and path of the input image are acquired (*filename1*);

— the file is read in the PGM format (*read_pgm*);

— the integer value of the pixels is converted in floating point number (*to_float*);

— to properly handle the user requirements (parallelism level, blur layers, and the

3. `github.com/preesm/preesm-apps/tree/master/SIFT`

various resolutions), counters have been instantiated (respectively: *counterPLevels*, *counterGpyrLayer* and *counterOctaveDownN*);

— in order to create the versions at various resolutions (octaves), the input image is upscaled once (*SPLIT_upsample2x* and *upsample2x*), and downscaled several times (*downsample2x1* and *downsample2xN*);

— regarding the blurring, the gaussian coefficient are calculated (*compute_gaussian_coefs*);

— the blurred images are computed for the upscaled image (*row_filter_transpose2x_1*, *BarrierTranspose2x_1*, *row_filter_transpose2x_2*, and *BarrierTranspose2x_2*) and the original one (*row_filter_transpose_1*, *BarrierTranspose_1*, *row_filter_transpose_2*, and *BarrierTranspose_2*);

— blur filtering is also applied to the downscaled images (*BarrierCounterGpyr*, *seq_blur1* and *seq_blurN*);

— all generated images are described by an array of 4 dimensions: octave (associated to the resolution), layer (representing the blur level), height, width;

— considering all resolutions and blur levels, an image pyramid is built in order to efficiently compute the difference of gaussian function in the next step (*MERGE_gpyr*);

— for each of these images, difference of gaussians (*ITERATOR_build_dog_pyr* and *build_dog_pyr*), and gradient and rotational metrics (*ITERATOR_build_grd_rot_pyr* and *build_grd_rot_pyr*) are computed;

— keypoints detection is performed (*ITERATOR_detect_keypoints* and *detect_keypoints*);

— an extraction phase refines the detected keypoints (*extract_descriptor*);

— keypoints deriving from the parallel evaluations are merged (*MERGE_keypoints*);

— an output image with the keypoints is stored as a PPM file (*draw_keypoints_to_ppm_file*);

— the keypoints are reported in a file (*export_keypoints_to_key_file*).

The structure of the srDAG changes along the reported scenarios. Indeed, the parallelism establishes the corresponding number of firings associated with the actors *to_float*, *upsample2x*, *downsample2x1*, *build_dog_pyr*, *build_grd_rot_pyr*, *detect_keypoints*

and *extract_descriptor*. On the other hand, some of the filtering actors (*BarrierTranspose2x_1*, *BarrierTranspose2x_2*, *BarrierTranspose_1*, *BarrierTranspose_2*, *seq_blur1*, and *seq_blurN*) are strictly correlated to the number of blur levels (the relation is equal to $nLayer+3$, except for the last one that depends on the number of the octaves too). Finally, the combination of the two chosen parameters leads to a specific number of instances for the tasks *row_filter_transpose2x_1*, *row_filter_transpose2x_2*, *row_filter_transpose_1*, and *row_filter_transpose_2*. The task graph shows a sufficient parallelism that can be effectively handled almost without the creation of further undesired activity.

5.2.2 Experimental Results

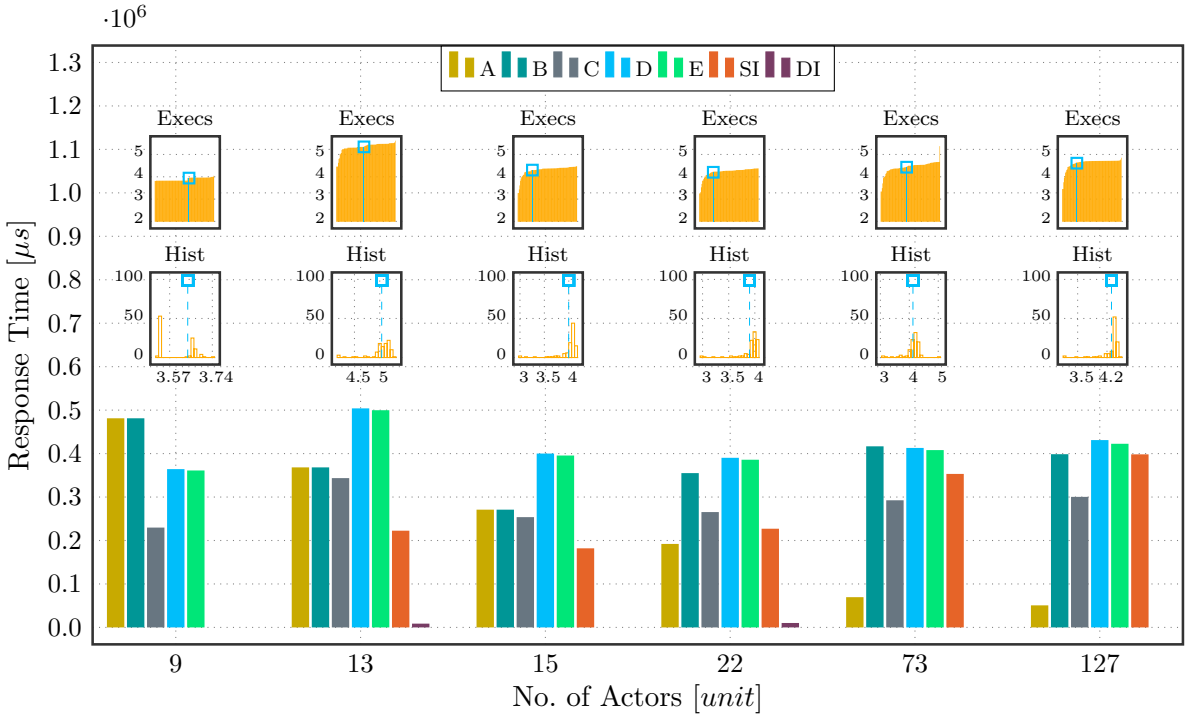
In this section, the latency evaluations associated with the levels presented in Section 3.2.2 for the use-case applications are shown. In particular, the task characterization for the a-priori information levels A and B has been obtained by the average of the execution times associated to the different types of PEs present in the target board. In Figure 5.3, a comparison of the knowledge levels presented in Section 3.2 in terms of response time estimation is proposed. Figure 5.4 shows the percentage error with respect to level D, expressed as average (Avg) and standard deviation (StD).

Video Stabilization

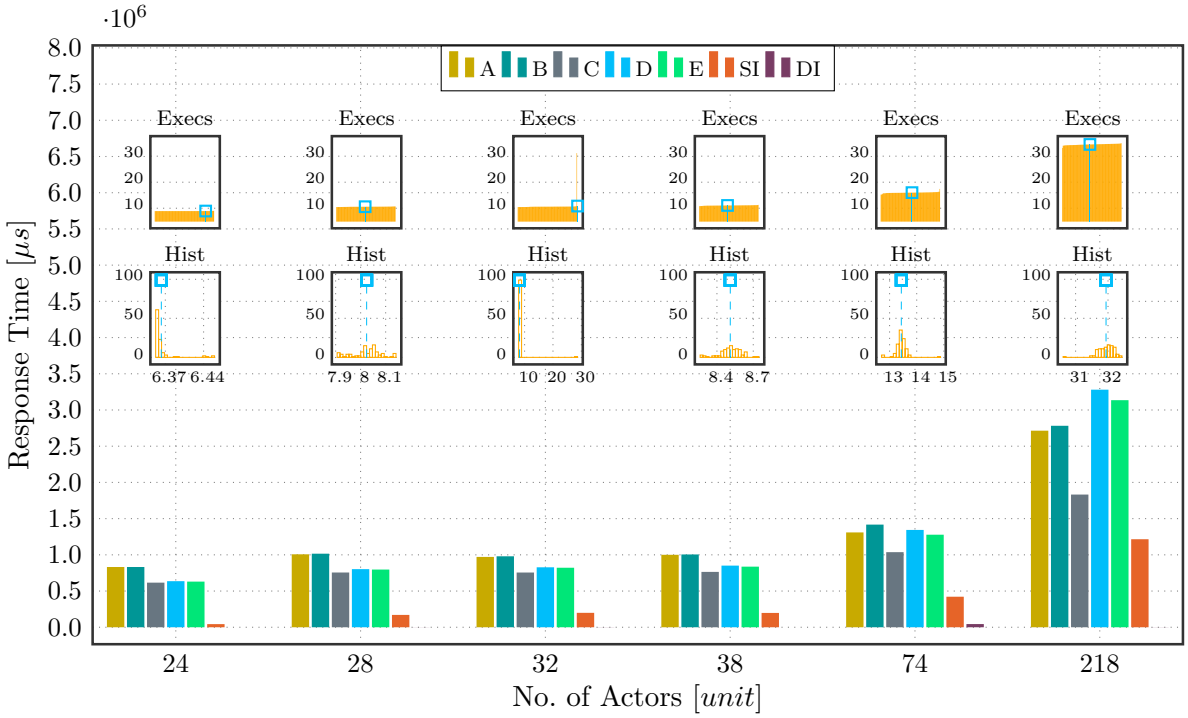
Starting from the first application, Figure 5.3A shows an inversely proportional trend between CP and real latency, as expected in presence of work domination. Indeed, with increasing number of tasks, the estimate values at level A decrease, while the interference grows. Compared to the level D (trace of the real latency) along all the scenarios, this leads to a poor accuracy depending only on the knowledge of the level A. As reported in Figure 5.4, its error in terms of average and standard deviation is -36.1% and 44.4% respectively. Regarding the other a-priori information levels (B and C), the estimates are more realistic since the exploitable parallelism in the application is higher than the number of the used PEs. Indeed, work domination is effective except in the first scenario (with 9 actors in total, but less than 8 with equivalent weights in parallel). Nevertheless, in these two levels, the error presents a mean value and a standard deviation equal to -2.0% and 22.2% (level B), and -33.1% and 2.9% (level C). These results derive from i) the

n	PiSDF Actor	Repetitions					
		2,1	2,2	2,3	2,4	4,2	4,4
1	filename1 (src)	1	1	1	1	1	1
2	read_pgm	1	1	1	1	1	1
3	SG1/to_float	2	2	2	2	4	4
4	SG1/counterPLevels (src*)	1	1	1	1	1	1
5	SG1/counterGpyrLayer (src*)	1	1	1	1	1	1
6	SG1/counterOctaveDownN (src*)	1	1	1	1	1	1
7	SG1/SPLIT_upsample2x	1	1	1	1	1	1
8	SG1/upsample2x	2	2	2	2	4	4
9	SG1/downsample2x1	2	2	2	2	4	4
10	SG1/downsample2xN	4	4	4	4	4	4
11	SG1/compute_gaussian_coefs (src*)	1	1	1	1	1	1
12	SG1/SG2/row_filter_transpose2x_1	8	10	12	14	20	28
13	SG1/SG2/BarrierTranspose2x_1	4	5	6	7	5	7
14	SG1/SG2/row_filter_transpose2x_2	8	10	12	14	20	28
15	SG1/SG2/BarrierTranspose2x_2	4	5	6	7	5	7
16	SG1/SG3/row_filter_transpose_1	8	10	12	14	20	28
17	SG1/SG3/BarrierTranspose_1	4	5	6	7	5	7
18	SG1/SG3/row_filter_transpose_2	8	10	12	14	20	28
19	SG1/SG3/BarrierTranspose_2	4	5	6	7	5	7
20	SG1/BarrierCounterGpyr	1	1	1	1	1	1
21	SG1/seq_blur1	4	5	6	7	5	7
22	SG1/seq_blurN	16	20	24	28	20	28
23	SG1/MERGE_gpyr	1	1	1	1	1	1
24	SG1/ITERATOR_build_dog_pyr (src*)	1	1	1	1	1	1
25	SG1/build_dog_pyr	2	2	2	2	4	4
26	SG1/ITERATOR_build_grd_rot_pyr (src*)	1	1	1	1	1	1
27	SG1/build_grd_rot_pyr	2	2	2	2	4	4
28	SG1/ITERATOR_detect_keypoints (src*)	1	1	1	1	1	1
29	SG1/detect_keypoints	2	2	2	2	4	4
30	SG1/extract_descriptor	2	2	2	2	4	4
31	SG1/MERGE_keypoints	1	1	1	1	1	1
32	draw_keypoints_to_ppm_file (snk)	1	1	1	1	1	1
33	export_keypoints_to_key_file (snk)	1	1	1	1	1	1
	No. of actor instances	101	118	135	152	172	222
	No. of edge instances	550	645	740	835	951	1229

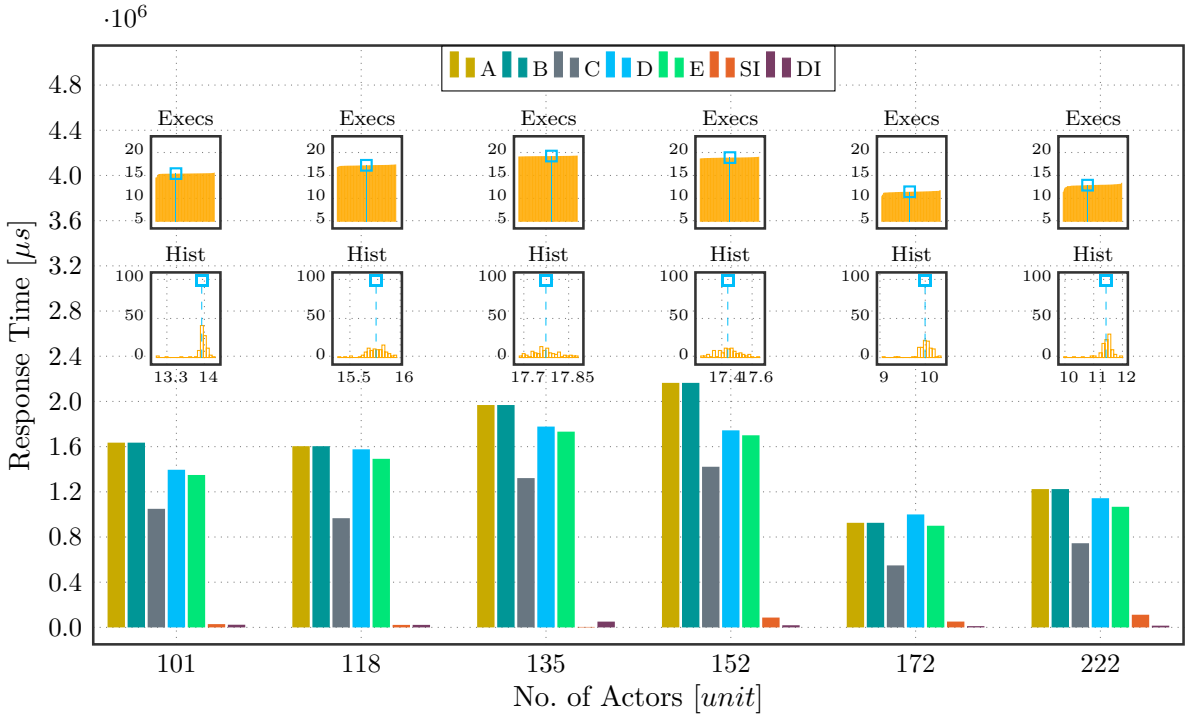
Table 5.7 – Actor repetitions in one DAG graph iteration in the chosen scenarios for the SIFT Point Computation application. Each scenario is represented by 2 parameters, with the following string: p1,p2; where, p1 is the parallelism level (parallelismLevel), p2 is the number of layers (nLayers). For this application, actors within a subgraph (SG) are preceded by SG1, and/or SG2, or SG3 (respectively named SIFT, Blur2x and Blur in the PiSDF). When the actor name is specified by (src) or (snk), this is a source actor or a sink actor of the graph respectively. A * is added to indicate an initialization actor unrelated to the input that can be evaluated as a source node in the I/O path analysis.



(A) Video Stabilization



(B) Stereo Matching



(C) SIFT Point Computation

Figure 5.3 – Response time estimate comparison with respect to the knowledge levels presented in Section 3.2. For each chosen scenario of the use-case applications, the selected execution (corresponding to level D) is highlighted among 100 measurements (in Execs) and their associated histogram (in Hist). Response time is represented in terms of $10^6[\mu s]$ and $10^5[\mu s]$ in the main plots and in the small ones (y-axis in Execs, and x-axis in Hist) respectively.

supposition of a homogeneous MPSoC which leads to a wide oscillation of the estimate around the average value (level B), and ii) from the evaluation which excludes scheduling delays due to non-ideal behaviors in execution which cause underestimated response times (level C). On the other hand, PathFinder (level E) achieves a high-accurated estimations, having an error with an average and a standard deviation respectively of -1.2% and 0.3%.

Stereo Matching

In the second application, the characteristics of parallelism and work lead to an incremental trend, when number of actors rises, of the activity associated to the CP and the interference, that can be seen in all the levels (as shown in Figure 5.3B). Regarding the error on the estimates depicted in Figure 5.4, the levels A and B suffer from a lack

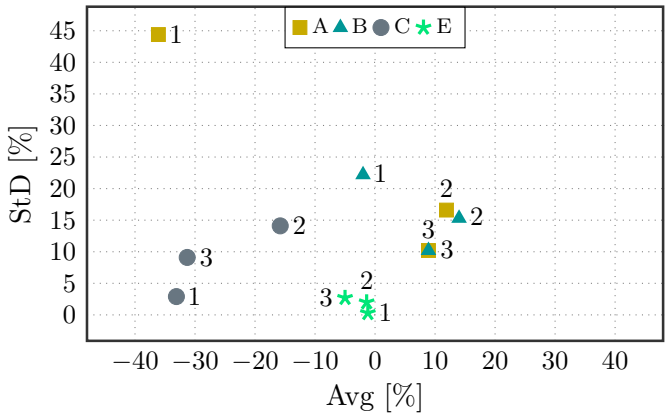


Figure 5.4 – Error comparison with respect to the reference level D. For each level, percentage error in terms of standard deviation (StD) and average (Avg) along the scenarios is reported for the 3 applications: Video Stabilization (1), Stereo Matching (2) and SIFT Point Computation (3).

of knowledge of architecture. Indeed, they show an error of 11.9% (16.6%) and 14.0% (15.3%) in terms of average (standard deviation) respectively. On the other hand, estimations based on level C show a decreasingly accurate estimate as the amount of work grows, with an mean error of -15.8% and a standard deviation of 14.1%. On the contrary, estimates obtained from the PathFinder (level E) follow the measured response times (Avg: 1.4%, StD: 2%). The good performances of PathFinder validate the heuristics composing the method. While not exploring all paths, PathFinder can still locate the causes of latency in the application.

SIFT Point Computation

As shown in Figure 5.3C, even if the presence of interferences does not dominate in this third application, PathFinder can reduce the estimate error with respect to the other levels (A, B and C). Indeed, PathFinder presents the following characteristics: Avg $= -5.0\%$ and StD $= 2.7\%$ (see Figure 5.4). On the contrary, results based on the a-priori levels (A, B and C) show an error with mean values of 8.9%, 8.9% and -31.3%, and deviations of 10.2%, 10.2% and 9.1% respectively.

5.2.3 Gantt Similarity Analysis

Nevertheless, since level C provides the highest execution knowledge based on a-priori information (i.e. information available before the execution), Figure 5.5 illustrates an

evaluation with respect to the metric Gantt Similarity (GS) defined in Section 3.2.3. In this comparison, the matching between the levels C and D has been considered. Figure 5.5 confirms the aspects described above for the estimate error. In Stereo Matching, since work and span are balanced, the accuracy of the estimation decreases with the number of actors, as the GS. In the other two applications respectively dominated by work and span, the values of GS does not present high variations. However, the similarity between the two Gantt charts remains low for all the applications, showing the limits of the response time estimates. These results show the complexity of the underlying problem, a real-life schedule being extremely different from levels A, B, and C sketches. Please note that GS cannot be drawn for PathFinder, as PathFinder constructs only the LLP, and not the complete Gantt.

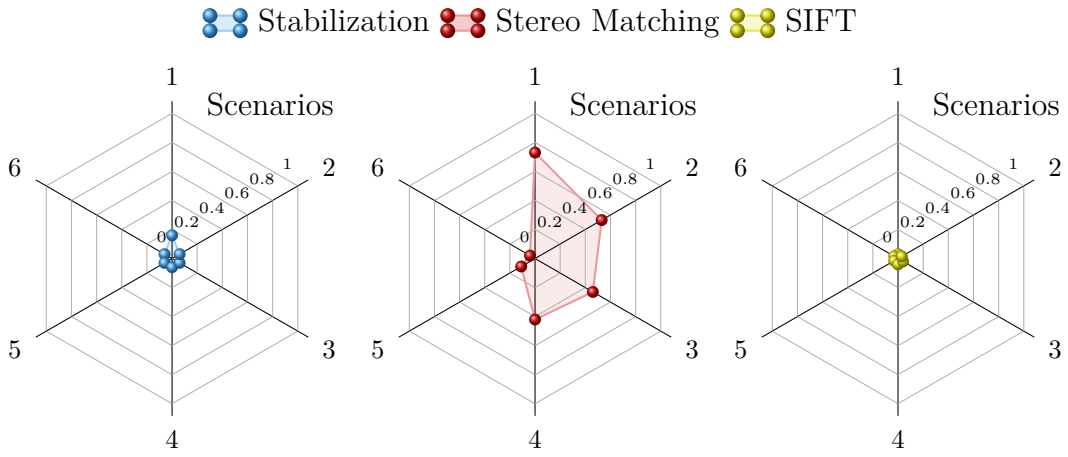


Figure 5.5 – Gantt Similarity evaluation for levels C and D by using the metric defined in the Equation (3.1).

5.2.4 Analysis of the Interferences

In this assessment, two types of interference have been evaluated. Indeed, the LLP analysis considers only the effects due to the scheduling and the dependencies, mentioned in Section 3.2.4. The choice fell on these two types, since, in the context of image/video processing, the actor latency dominates the communication latency. Otherwise the system would be inefficient, spending its time communicating data and not processing. Figure 5.3 also shows the subdivision of the interference for the scenarios of the Section 5.2.2 (SI and DI indicate scheduling and dependency interferences respectively). The interference

of Video Stabilization grows with the amount of work of the actors and is mainly due to scheduling. Such an analysis is actually a method to determine that the application is work-dominated. This derives from the strategy applied by PREESM, that favors the mapping on the faster PEs until the parallelism does not give any advantage. On the contrary in SIFT Point Computation, the dependency interference affects the CP until the last scenarios. However, the CP contribution mainly determines the response time, since the span dominates the execution. In Stereo Matching, both scheduling and dependency interferences grow with work. In the 5-th scenario the dependency interference starts to have a significant weight. As expected by an application in which coexist work and span effects, the combination of both types leads to an increase with the number of the actors.

5.3 Chapter Remarks and Future Works

PathFinder aims at explaining factors that determine system execution latency. The design of PathFinder design flow provides novel latency analyses for parametric applications. In addition, a new metric called Gantt Similarity that guides the response time estimation process has been introduced. The method assessment on three non-toy applications shows the difficulties in predicting and understanding response time depending on a-priori information. On the other hand, PathFinder provides execution awareness based on applications described as an srDAG as well as on monitored execution times. PathFinder generates high-accuracy latency estimates based on statistical execution behaviors.

However, the scenarios evaluated in the assessment belong to an application domain (image/video processing) in which task processing dominates the communication costs. Thus, out-of-domain scenarios with important communication costs have been neglected in the assessment. Moreover, the detection of the interferences could be extended in order to cover the combinations of the presented types, such as the following ones:

— *scheduling affected by dependency*: such interference appears when an actor that interferes with CP through scheduling interference has to wait for a token from other tasks, which affects its execution with a dependency interference;

— *dependency affected by scheduling*: such interference appears when an actor that interferes with depending interference has to wait for the execution of other tasks, which affect its execution with a scheduling interference.

These second-order interferences are not evaluated yet by PathFinder while, in some real cases, they do have a non-negligible impact, especially when the CP estimate is underperforming. The current version of the PathFinder LLP detection does consider communication costs as embedded inside tasks costs. This hypothesis is valid in the case of shared memory MPSoCs because cache accesses and cache management operations (write-back, invalidate) are entangled with processing operations. When considering architectures with distributed memories, such as multiple networked systems, communications will need to be considered separately.

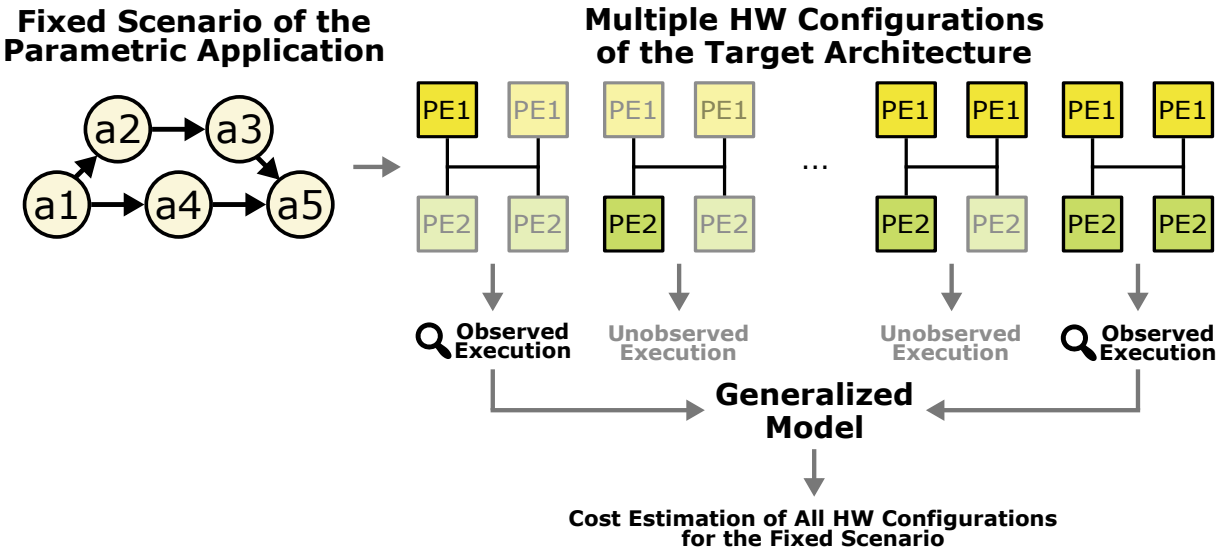
In addition, timing analyses are especially useful also in the context of runtime management. Indeed, for the dynamic handling of task scheduling and mapping, runtime managers rely on profiling of the system execution in order to apply the adaptation strategies [Sin+17]. PathFinder provides response time analysis based on specific statistical metrics, suitable for parametric applications. Future works can consider an improvement with an a-priori version, in order to favor runtime adaptation based on LLP analysis.

Chapter 6

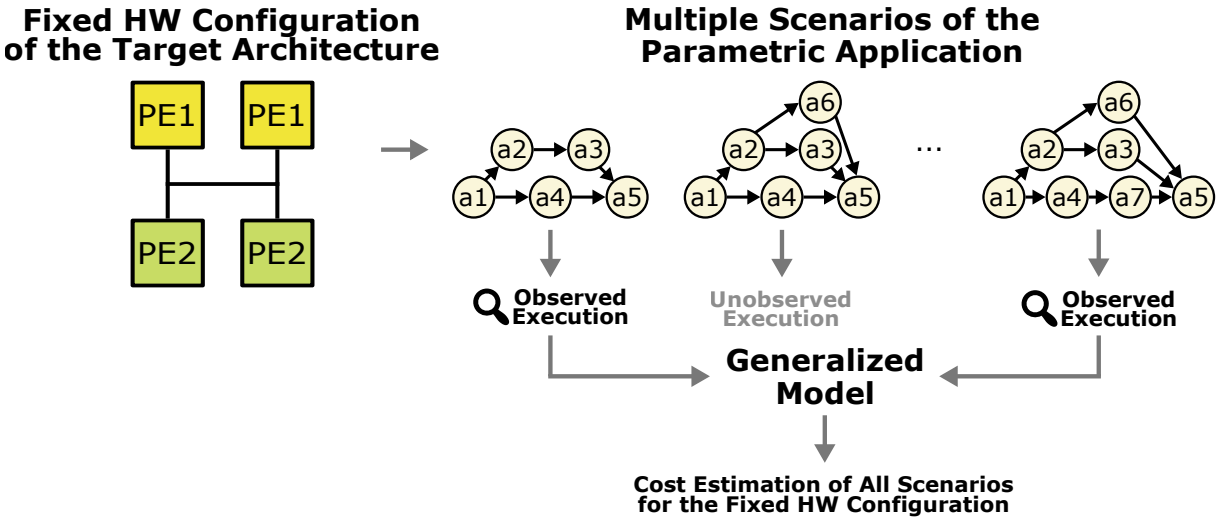
Contribution 3: Response time estimation through LLP Activity and a Model of Architecture

In this chapter, a flow aimed at building, training and assessing a Model of Architecture (MoA) to access response time is proposed. The purpose of the flow is to evaluate a novel top-down strategy for reducing complexity in estimating system execution latency of a parametric application described as a DF specification, and mapped upon a heterogeneous system. More generally, the question is on whether we can generalize a model to unobserved cases. Indeed, mapping tasks of an application on specific sets of PEs of the target architecture (HW configurations) leads to different execution costs in terms of specific KPIs, since performance and scheduling of tasks are modified depending on the number and type of selected PEs. On the other hand, even fixing the HW configuration, values of parameters used to perform a particular functionality or its different operating modes (scenarios) give rise to distinct execution costs (e.g. due to changes of the exploitable parallelism level), as already seen in Chapter 5 regarding response time. A generalized model is capable of providing cost estimation for all desired HW configurations and scenarios, basing on a few observed cases used for training the model (see Figure 6.1). Modifying HW configurations and scenarios makes it possible to separate functional and architectural sides while identifying inter-relations between them. Figure 6.1A illustrates the estimation capability of a generalized model when a scenario-based application is mapped on a target architecture evaluated in its different HW configurations. Instead, Figure 6.1B shows how a generalized model can estimate execution costs of multiple scenarios upon a HW configuration.

As also introduced in Section 3.3, MoAs have been used as generalized models with promising results, modeling an MPSoC with high fidelity in energy estimation [Pel+18]. MoAs provide reusable tools for specifying KPI-based models that are abstract, application-



(A) Generalizing a model for all HW configurations given a fixed application scenario.



(B) Generalizing a model for all application scenarios given a fixed HW configuration.

Figure 6.1 – Model generalization for multiple HW configurations and application scenarios.

agnostic (as much as possible), and reproducible. The foundations of an MoA are represented by the concepts of *activity* and *cost function*. Activity consists of a representation/model of application-side sources that determine the execution cost on the targeted architecture with respect to a desired KPI. Any other source not in the given activity does not impact the KPI-based cost. A cost function specified at MoA-side expresses the relation between activity and its cost when this is mapped onto a specific element of the

targeted architecture. An MoA is represented by a set of cost functions in equal number with respect to the abstract elements describing the target architecture, whether they are associated with processing or communication. An example of more detailed definitions of activity and cost function will be provided in Section 6.1.

Although the promising results for energy modeling, the context of timing analysis implies a more complex characterization of the activity that determines the metric on which the MoA is based. Indeed, the additive nature of energy makes all processing and communication in the workload to enter activity. For this purpose, PathFinder can be exploited in order to provide an activity efficient for response time analysis. Indeed, as demonstrated in Section 5.2, LLP represents a set of execution time contributions that properly fits the response time of the whole application. The question is now: can we generalize the output of PathFinder to unobserved cases?

This chapter is organized as follows. In Section 6.1, a definition of the MoA linked to the activity provided by the PathFinder (such as LLP) is presented. Section 6.2 shows how the amount of information involved in the model building may affect in terms of accuracy and fidelity. The effects of these degrees of freedom are examined more in details with respect to the training of the model in Section 6.3. In Section 6.4, the design flow considered for the training and evaluation is described. The assessment of the model is proposed in Section 6.5. Comments related to this chapter are presented in Section 6.6.

6.1 Using the LSLA MoA for LLP-based Activity

If the LLP is well built by PathFinder, the response time of the application is obtained by summing the effect of each LLP element. Thus, combining the notion of LLP and a linear MoA is a promising direction for decomposing the problem of response time model generalization. When an activity is mapped onto an MoA, the cost due to its execution can be estimated, either at design time if everything is known on this activity, or at runtime if activity is *discovered* along execution. The MoA considered in this thesis is the Linear System-Level Architecture Model (LSLA) proposed in [Pel+18] (see Figure 6.2), which corresponds to an undirected graph ($\Lambda = (P, C, L, cost, \lambda)$) respectively composed of sets of PEs (P), Communication Nodes (CNs) (C), connection links (L), cost functions ($cost$) and coefficients of the cost ratio between processing and communication (λ). A constraint for the model to be an MoA is that, in order to be reproducible, it must specify cost computation (the cost being the KPI of interest). The amount of "pressure" put on

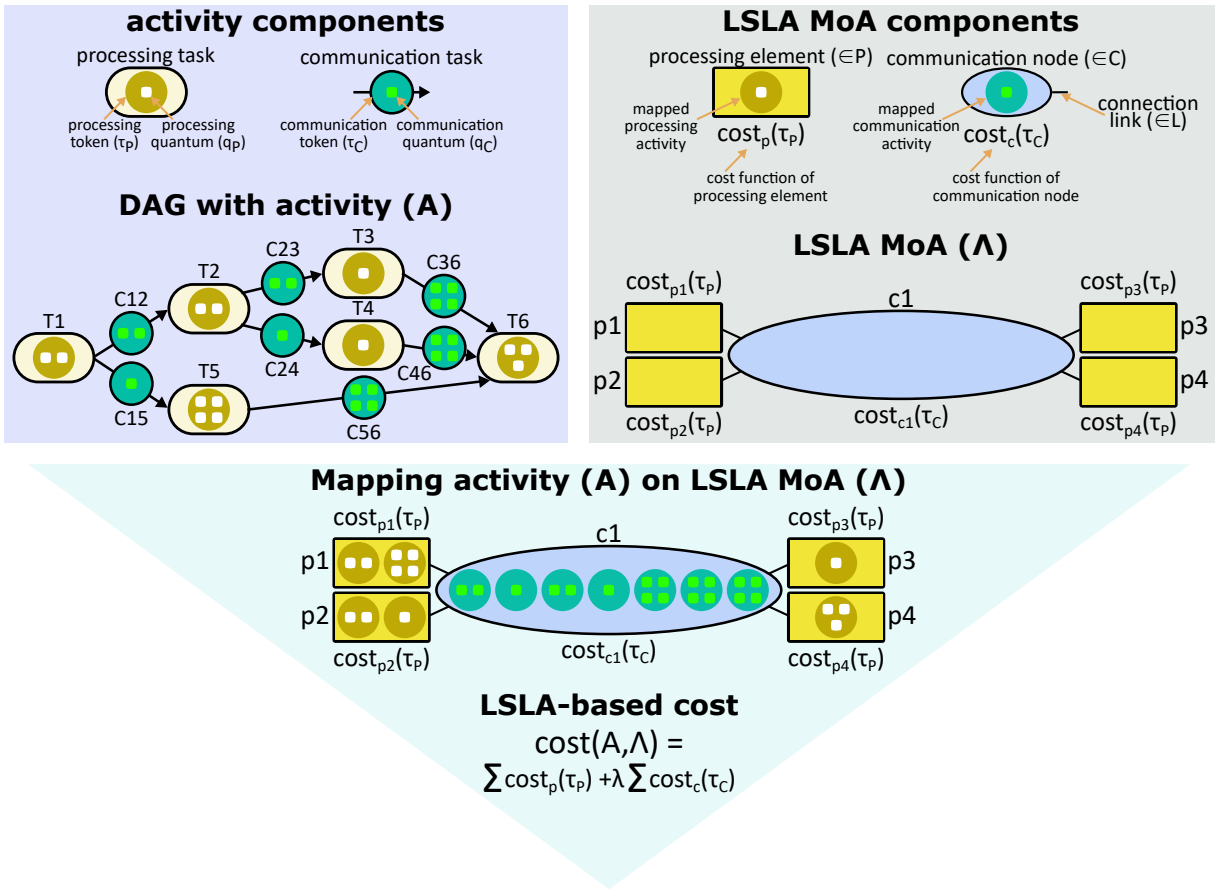


Figure 6.2 – Overview of the Y-chart-based estimation flow by using LSLA MoA.

hardware by the application is abstracted by a notion of *activity* composed of *tokens*. A token itself is composed of a natural number of non-divisible *quanta*. In particular, a processing (resp. communication) token τ_P (τ_C) must mapped onto a PE p (CN c). When a token τ is mapped onto a node n (a PE or a CN), its cost in LSLA is computed by a function defined as:

$$cost_n = \alpha_n \cdot size(\tau_n) + \beta_n = \alpha_n s_n + \beta_n \tag{6.1}$$

where α_n and $\beta_n \in \mathbb{R}$. α_n (β_n) represents the fixed cost (overhead) of a quantum q (token τ_n) executed on n (potentially in the presence of co-running interference). The idea behind the two-levels construct with tokens and quanta is for the activity to include two levels of granularity (typically function/cycle in processing and message/byte in communication). As will be proposed in Section 6.3, α_n, β_n and the cost unit ν can be expressed in

time/quantum, *time/token*, and *time* respectively when targeting response time. While in the energy case, α_n, β_n and ν have been respectively defined in *energy/quantum*, *energy/token*, and *energy*. However, the main difference between energy and response time lies in the cost computation of the model. While the the cost of the execution of the energy-related activity A onto the LSLA MoA Λ with $size(P)$ PEs and $size(C)$ CNs is:

$$cost(A, \Lambda) = [\sum_{\tau_p}^{T_p} cost_p] + \lambda \cdot [\sum_{\tau_c}^{T_c} cost_c] \tag{6.2}$$

the cost of the counterpart LLP-based activity A onto the LSLA MoA Λ can be estimated as follows:

$$cost(A, \Lambda) = [\sum_{\tau_p}^{T_p \subset LLP} cost_p] + \lambda \cdot [\sum_{\tau_c}^{T_c \subset LLP} cost_c] \tag{6.3}$$

with (in both Equations 6.2 and 6.3)

$$\begin{aligned} cost_p &= \alpha_p s_p + \beta_p \\ cost_c &= \alpha_c s_c + \beta_c \end{aligned} \tag{6.4}$$

where $\lambda \in \mathbb{R}$ is the Lagrangian coefficient setting the relative weight between communication-related and processing-related costs per quantum, and $p \leq size(P)$ ($c \leq size(C)$), with $size(P)$ ($size(C)$) the number of PEs (CNs) present in the system. p (c) corresponds to the index of a specific PE (CN) on which tokens are mapped. These represent the activity as contributions belonging to the LLP. However, this implies that a set of tasks present in the DAG determines a cost in terms of response time. In addition, LLP depends on the mapping solutions of a determined DAG, which in turn derives from a single operating mode of the parametric application. So if we want to generalize the MoA to all or a part of parameter values in a target application, the LLP needs to either be relatively stable, or recomputed based on application parameters. In fact, each application scenario may involve a different number of tasks and therefore lead to a distinct LLP. For these reasons, the total cost defined by Equation (6.3) needs to be evaluated for each desired mapping and corresponds to the response times associated with a determined LLPs. This enables estimates with respect to specific mapping configurations, which consider only the PEs actually involved in the execution of the activity associated with the LLPs. In particular,

since an MoA can be tailored for different statistical metrics for the timing analyses (such as the maximum or the average), α_n and β_n can be derived for each desired behavior. Furthermore, in contexts where application activity is mainly due to processing, so that it is predominant with respect to communication, it may be convenient to consider a simplified model, ignoring communication costs, as follows:

$$cost(A, \Lambda) \approx \sum_{\tau_p}^{T_p \subset LLP} cost_p \tag{6.5}$$

6.2 The Long Road Towards the Fully Generalized MoA

A fully generalized MoA would be applicable to all functionalities and HW configurations for a given platform family (e.g. multi-ARM). In order to use an MoA, some assumptions related to the model building need to be clarified. Indeed, depending on the choice of the activity and the procedure used for obtaining model parameters (α_n and β_n), cost functions can present a certain degree of accuracy making it suitable to specific domains. In general, α_n and β_n are obtained as a function of the cost-activity pairs (e.g. through linear regression). This fact implies a strict relation between these data and the domain of accuracy of the model, which can concern specific scenarios, applications, and HW setups. For this reason, the steps aimed at obtaining a model with a large domain of accuracy are explained in the following sections. In this work, generalization is achieved incrementally, first looked for on new mappings to the same platform, then to new scenarios of the same application and finally to new forms of application.

Single Mapping Information (SMI)

The simplest model can be based on the information about the LLP of a single mapping solution of an application scenario. This is actually a degenerated case with no generalization where only "curve fitting" is tested. This leads to unitary cost functions expressed as follows:

$$cost_{n,i} = \alpha_n \cdot size(\tau_{n,i}) + \beta_n = \alpha_n s_{n,i} + \beta_n \tag{6.6}$$

which corresponds to Equation (6.1) with the dependency on the LLP activity highlighted through the index i. The latter is associated with the i-th task present in the LLP of the considered mapping for the scenario. A model based on Equation (6.6) can be used in identifying the cost of a predetermined mapping solution of a single scenario for a specific application. Indeed, each scenario is represented by a unique DAG, which can be mapped on a certain number of PEs present in the target architecture. Since the cost function is derived from the knowledge of a single mapping configuration, response time is likely to be roughly estimated. However, re-mapping LLP tasks to PEs with different α_n and β_n is a first method to generalize the model to a different mapping.

Multiple Mappings Information (MMI)

In order to achieve a more accurate estimation, the model building can take into account m mappings of the same scenario. With such a strategy, for each task the couple of the variables cost and activity depends also on a selected number of mapping solutions. The fixed DAG can for instance be mapped to different sets of PEs. Accordingly, the linear parameters of the cost functions are extracted considering also this dependency on the mappings, and can be expressed as follows:

$$cost_{n,i,m} = \alpha_n \cdot size(\tau_{n,i,m}) + \beta_n = \alpha_n s_{n,i,m} + \beta_n \tag{6.7}$$

where the computation of α_n and β_n can provide for more than one step. For instance, a linear regression with respect to the m-th set of cost-activity pairs, and therefore an average operation on the m values of α_n and β_n.

Such a model is aimed at estimating response time for the activity associated with further mappings of the fixed DAG, which are different from the m used in model building.

Multiple Scenarios Information (MSI)

The purpose of the model can be to estimate the system execution latency for different scenarios of the same application. In this case, different DAGs can be involved in building of the parameters of cost functions. As seen in Section 5.2, tasks of different scenarios often differ from each other in terms of number, duration, dependencies with other tasks. This leads to the evaluation of this information in the construction of the model. Therefore, in the cost functions should appear the dependency on s selected scenarios of the variables

related to cost and activity of the tasks:

$$cost_{n,i,m,s} = \alpha_n \cdot size(\tau_{n,i,m,s}) + \beta_n = \alpha_n s_{n,i,m,s} + \beta_n \tag{6.8}$$

Multiple Applications Information (MAI)

In order to build an MoA capable to provide estimates for different applications, α_n and β_n parameters can be evaluated in their related scenarios. Indeed, cost functions should take into account such information in the choice of the cost and activity:

$$cost_{n,i,m,s,a} = \alpha_n \cdot size(\tau_{n,i,m,s,a}) + \beta_n = \alpha_n s_{n,i,m,s,a} + \beta_n \tag{6.9}$$

Involved Information	Mappings	Scenarios	Applications
SMI	1	1	1
MMI	m	1	1
MSI	m	s	1
MAI	m	s	a

Table 6.1 – Overview of the information involved in building of the cost functions.

6.3 Training an MoA from System Platform Measurements

Different solutions, given by the activity definition and the computation of the cost function parameters, make the MoA an enabler for DSE methods with a high level of elasticity. Indeed, an MoA-based method is capable of trading off between fidelity and estimation complexity (as shown in Section 6.5). In particular, the training leading to parameter values of the function costs, defined in Equation (6.1), that are the basis of the MoA, can be performed in many different ways.

6.3.1 Example of choice of the activity

Considering an LLP-based activity for MoA training, single token cost can be defined as follows:

$$size(\tau_n) = s_n = w_n \cdot Char\_cost_n = \frac{LLP\_cost_n}{Real\_Exec\_cost_n} \cdot Char\_cost_n \tag{6.10}$$

where $w_n \in \mathbb{R}^+ : w_n \leq 1$ represents the weight of the token (mapped on the n-th PE) relative to the execution (selected by the PathFinder) with respect to its characterized value ($Char\_cost_n$) taken as a reference. In this section, this weight is obtained from the ratio of costs in terms of execution time that the task associated with the token presents in the LLP (LLP_cost_n) and in the real execution ($Real\_Exec\_cost_n$). In particular, activity depends on the information used to build the cost function, as seen for (6.6), (6.7), (6.8) and (6.9). This leads to explicitly represent activity with respect to the information used in the construction, for the levels SMI, MMI, MSI, MAI respectively as follows:

$$s_{n,i} = \frac{LLP\_cost_{n,i}}{Real\_Exec\_cost_{n,i}} \cdot Char\_cost_{n,i} \tag{6.11}$$

$$s_{n,i,m} = \frac{LLP\_cost_{n,i,m}}{Real\_Exec\_cost_{n,i,m}} \cdot Char\_cost_{n,i,m=m_{ref}} \tag{6.12}$$

$$s_{n,i,m,s} = \frac{LLP\_cost_{n,i,m,s}}{Real\_Exec\_cost_{n,i,m,s}} \cdot Char\_cost_{n,i,m=m_{ref},s} \tag{6.13}$$

$$s_{n,i,m,s,a} = \frac{LLP\_cost_{n,i,m,s,a}}{Real\_Exec\_cost_{n,i,m,s,a}} \cdot Char\_cost_{n,i,m=m_{ref},s,a} \tag{6.14}$$

where the weight w is respectively related to the LLP and real execution costs of the i-th task of: i) the single mapping of chosen scenario, ii) the m mappings of chosen scenario, iii) the m mappings of s scenario of the chosen application, and iv) the m mappings of s scenario of the a applications. On the other hand, i) a single mapping in (6.11) or one of the m mappings: ii) of the same scenario in (6.12), iii) for each of the s scenarios in (6.13), iv) for each of the s scenarios of the a applications in (6.14), is selected as a reference of

the characterized cost of the tasks.

6.3.2 A method for choosing the mappings

The complexity of the timing analysis is mostly due to the large number of possible combinations of the several variables involved. For instance, the number of considered mappings can lead to an explosion of the analysis. Indeed, if within the application graph and the architecture are respectively present c actors and K PEs, the number of their mapping combinations are equal to K^c. Thus, considering 10 actors mapped onto 8 PEs, more than a billion of different mappings could be analyzed. In analysis that considers a pruned space of actors and of PEs is necessary. For these reasons, in order to face complexity, the proposed training considers a number of mapping solutions that are dependent only on the type t and on the number K of the PEs. When considering a platform, with the number of PE types lower than the number of PEs (i.e. with more than one PE of each type), the mappings with most interest for the training set are the ones that differ from each other in the number of PEs of at least one type. In particular, the training relies on the scheduling and mapping algorithm of the compilation tool in order to find unconstrained solutions regardless of number of actors, as considered in PathFinder. In this case, the number of mappings depending only on the type t of each PE and the number K of the PEs are:

$$M = \sum_{k=1}^{K} \frac{(t+k-1)!}{k!(t-1)!} \tag{6.15}$$

where k is the number of currently active PEs, on which tokens can be mapped. Thus, for each architectural configuration corresponds a values of the index $k \leq K$. For example, with 8 PEs ($K = 8$) of 2 different types ($t = 2$), Equation (6.15) becomes $\sum_{k=1}^{8}(k+1) = 44$ combinations of different mappings to be evaluated. Indeed, for the k-th configuration, there are $k + 1$ combinations of different PE types to have k of them that are active. However, considering to have an equal number of PEs for each type present in the architecture (in our case 4), the number of differing mapping solutions decreases to 24. In fact, Equation (6.15) becomes:

$$M = \sum_{k=1}^{K/2}(k+1) + \sum_{k=K/2+1}^{K}(K-k+1) \tag{6.16}$$

Such strategy leads to measure $m < M$ mappings in the training of the model. The other $m' = M - m$ mapping solutions can be used as a test set to assess the model (since they serve as unobserved cases). Table 6.2 reports the M mappings derived from Equation (6.16).

Mapping Index	**No. of Active PEs**	**No. of Active t_1-type PEs**	**No. of Active t_2-type PEs**
1	1	1	0
2	1	0	1
3	2	2	0
4	2	1	1
5	2	0	2
6	3	3	0
7	3	2	1
8	3	1	2
9	3	0	3
10	4	4	0
11	4	3	1
12	4	2	2
13	4	1	3
14	4	0	4
15	5	4	1
16	5	3	2
17	5	2	3
18	5	1	4
19	6	4	2
20	6	3	3
21	6	2	4
22	7	4	3
23	7	3	4
24	8	4	4

Table 6.2 – HW configuration with respect to the mapping solutions given by the (6.16).

6.4 Estimation Flow through MoA with LLP-based Activity

Figure 6.3 illustrates the design flow used to train and assess the LSLA MoA with LLP-based activity. This flow starts from the selection of i) s application scenarios, described through the specifications given by the DF MoC with a determined set of parameters,

and ii) the architecture to be modeled, which embeds K PEs configured to have k of them active. Among all potential mappings, M desired solutions are chosen, e.g. using Equation (6.16). These are analyzed by PathFinder in order to obtain as many LLPs. In this flow, LLPs represents the activity of a single srDAG related to a specific scenario. Therefore, a certain number m of LLPs is used in the training of the cost functions. As an output of this phase, m sets of (α,β) pairs associated with the t types of PEs present in the architecture model are provided. Leveraging on these sets, a final version of the cost functions is defined depending on the assumptions described in Section 6.2. At this point, the cost functions complete the whole model, which is previously expressed by Equation (6.5). Such model is ready to be assessed with the other $m' = M - m$ LLPs in order to evaluate its accuracy and fidelity in a generalized mapping case and under the hypothesis that the LLP is well known, by using Equations (3.2) and (3.3).

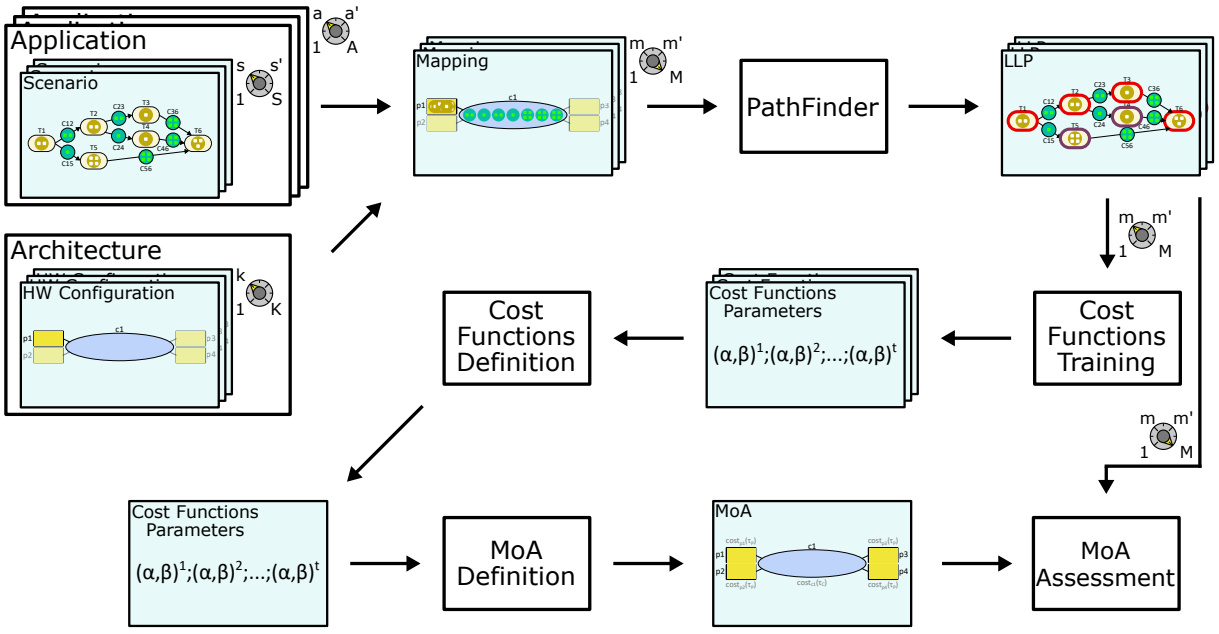


Figure 6.3 – Response time estimation flow through LSLA MoA

6.5 Assessment of the Method

In this section, a first assessment of the proposed MoA-based method for application with processing that dominates communication is proposed. The use case corresponds to the 3rd scenario of the Stereo Matching application proposed in Section 5.2.2. In the

same way, this has been mapped onto an ODROID-XU3 board [Har], which is composed of $K = 8$ ARM cores (4 CORTEX-A7 and 4 CORTEX-A15, $t = 2$) as specified previously. The scenario has been evaluated in $M = 24$ different mapping solutions, as proposed in Equation (6.16). The model is evaluated based on single and multiple mappings of the unique scenario (SMI and MMI respectively), as described in Section 6.2. The model obtained by the single mapping considers one mapping per PE type. In this case, the chosen mapping solution ($m = 1$) is associated with the execution of the scenario on only one active PE for each PE type (mapping indexes 1 and 2 of Table 6.2). On the other hand, the models based on multiple mappings per PE type are related to the solutions in which PEs of a unique type are solely active (as in Table 6.2 for the mapping indexes: 1, 3, 6 and 10 for the type CORTEX-A7; and 2, 5, 9 and 14 for the type CORTEX-A15). Table 6.3 reports the values obtained for the parameters of the cost functions for both models, which have been trained by using Equations (6.11) and (6.12) respectively.

Model	**Training**					**CORTEX-A7**		**CORTEX-A15**	
	K	t	M	m	m'	α	β	α	β
MoA (SMI)	8	2	24	2	22	1.63	53437.97	1.02	-966.32
MoA (MMI)	8	2	24	8	16	1.56	29946.72	1.01	536.70

Table 6.3 – Parameters of the cost functions for the models based on the information of single and multiple mappings per PE type.

Figure 6.4 shows the estimation results compared to the real response times and the LLP of the selected scenario. The reported executions correspond to the average ones derived from the PathFinder analysis. In order to explain the properties of the model estimates, Table 6.4 reports the results in term of fidelity and percentage error. Starting from the input of the models, LLP presents a high accuracy with an mean error of -2.04% and a standard deviation of 1.39%. With respect to the fidelity metrics, LLP achieves almost the perfect correlation considering the Spearman's rank correlation coefficient ($FM_\rho = 0.98$), and a high fidelity for the Kendall's tau ($FM_\tau = 0.92$). On the other hand, the training of the model with single and multiple mapping solutions leads to a clear reduction depending on the used metric: FM_ρ (0.94 and 0.95) and FM_τ (0.85 and 0.83). Although the model based on multiple mappings is more refined, this is not clearly shown in terms of fidelity as in those of accuracy. In fact, its estimation error exhibits better results in average (-3.17% vs 3.97%) and standard deviation (5.51% vs 11.25%).

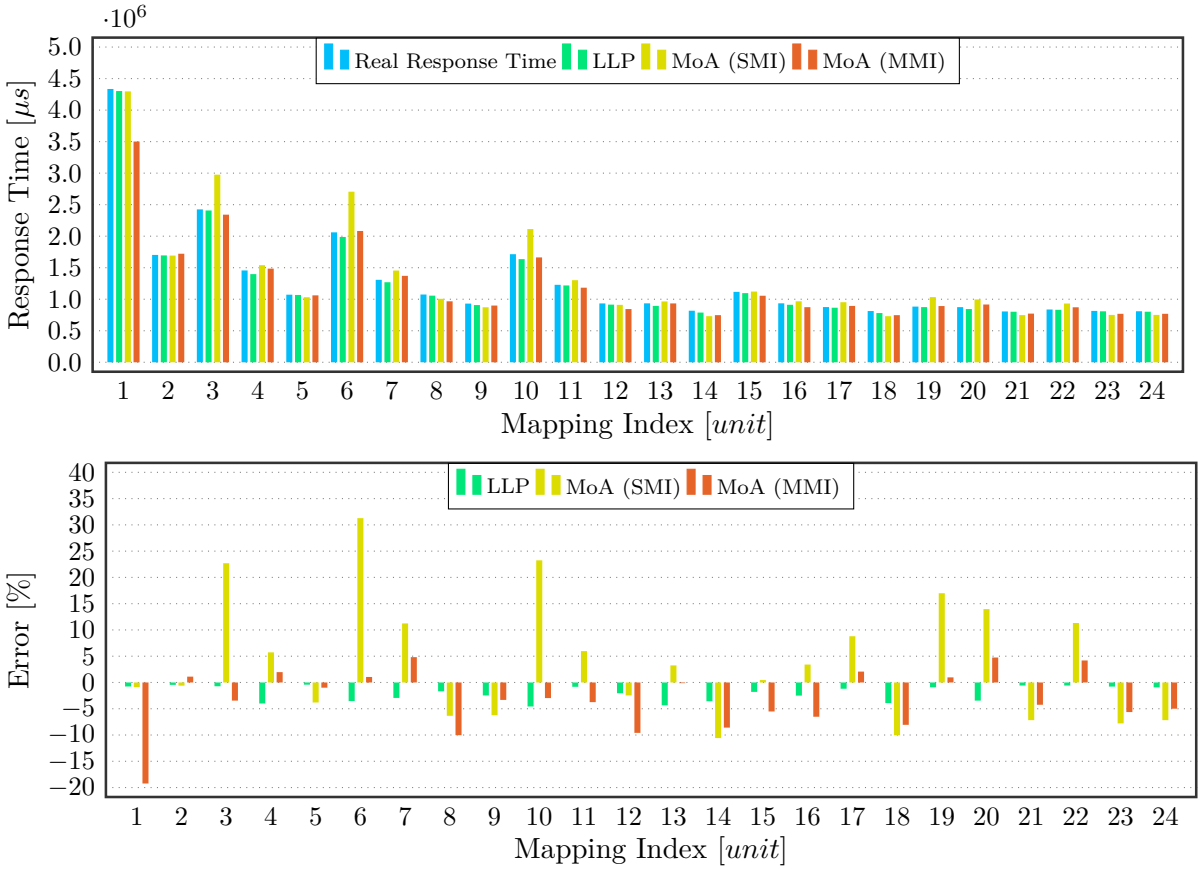


Figure 6.4 – Comparison of the real response time with the estimation provided by the LLP, and MoAs based on single and multiple mapping information (SMI and MMI respectively).

Model	**Fidelity Metric**						**Error [%]**			
	FM_ρ	FM_τ	n	n_c	n_d	$\sum_{i=1}^{n} r_i^2$	min	max	avg	std
MoA (SMI)	0.94	0.85	24	255	21	136	-10.58	31.28	3.97	11.25
MoA (MMI)	0.95	0.83	24	252	24	124	-19.23	4.82	-3.17	5.51
LLP	0.98	0.92	24	265	11	36	-4.57	-0.41	-2.04	1.39

Table 6.4 – Comparison of the models in terms of Fidelity Metric, by using the (3.2) and (3.3), and estimated error compared to the real latency.

6.6 Chapter Remarks

In this chapter, an method aimed at estimating response time based on LLP and the LSLA MoA has been proposed. Compared to the state-of-the-art approaches, MoAs represent a promising alternative in order to increase the applicability of timing analysis to

a large family of systems. Especially in contexts that require the execution of parametric applications on heterogeneous systems, MoAs can be used to support reconfiguration and adaptation management. Indeed, MoAs can leverage on a simple characterization of the factors that determine an execution KPI such as the system execution latency. This fact enables more tractable KPI evaluations, in particular when the choice of the activity leads to linear MoAs. Nevertheless, finding a proper activity representing response time of multi-scenario functionalities is not a trivial operation. In order to present a first assessment of the proposed MoA, activity has been modeled leveraging on the LLP provided by the PathFinder. Although this choice may appear advantageous in building cost functions, in current work, it requires to be obtained for each desired mapping. Therefore, the LLP can be considered useful in the verification of accuracy and fidelity of an MoA based on this activity, but an important remaining open question is on how to also generalize the LLP to unseen cases.

In addition, a reduction of the complexity, due to the analysis of more HW/SW details, can be achieved at the cost of a loss of accuracy in the estimation. Indeed, as seen in the assessment, the estimation error can raise compared to the LLP model provided as an input. However, the MoA is capable of preserving a reasonable degree of fidelity with respect to the Spearman's and Kendall's metrics. However, these sets of results are certainly preliminary, therefore the work associated with estimating response time through an MoA needs to be refined before applying the strategy on a hybrid HW/SW system.

Chapter 7

Conclusion

Combining heterogeneous devices in MPSoCs in order to increase flexibility and efficiency in embedded and cyber-physical systems increasingly represents an inevitable design choice. In particular, a balanced trade-off between specialization and flexibility is the composition of a system based on general-purpose and CGR PEs. In this context, a functionality with multiple operating modes can be described with generic SW specifications, and accelerated with low reconfiguration overhead for its computationally intensive workloads. However, mapping such a functionality upon a set of specific PEs requires the use of a Model of Computation which provides a modular description and defines the execution rules of its parametric tasks. For this purpose, DF MoCs represent a suitable solution for a design based on both HW and SW implementations of signal/streams of data. Nevertheless, in order to estimate response time of parametric functionalities mapped on heterogeneous MPSoCs, state-of-the-art approaches request complex timing analyses that are rarely applicable in practice because details of the hardware are unknown and/or under NDA. In particular, a simplification of architecture models aimed at supporting live adaptation strategies is required.

For these reasons, this thesis work has been focused on an new method that pinpoints from platform measurements the factors that determine response time. In particular, the contributions of the thesis have proposed and assessed the following elements (see Figure 7.1):

— Chapter 4 - Contribution 1: The DataFlow-Functional HLS, created from the combination of the CAPH functional programming DF-based HW generation and the MDC reconfigurable HW datapath management tool, makes it possible to design a system with a parametric functionality or, more specifically, with one of its parametric tasks implemented as a CGR IP. Compared to the state-of-the-art approaches, DFF HLS leads to i) early predictability in terms of IP-level response time and ii) functionality-oriented solution synthesizable for different platforms (vendor-agnostic FPGAs and ASIC). Co-processing with other general-purpose processors, the gener-

ated IP can be integrated in an MPSoC by using the proposed adaptation framework for dynamically managing parametric functionalities implemented through HW and SW specifications based on a dataflow representation of the application.

— Chapter 5 - Contribution 2: The PathFinder tool is proposed that aims at understanding factors that determines system-level response time for parametric functionalities mapped on heterogeneous MPSoCs. Compared to the traditional approaches, PF proposes a different point of view for identifying timing activity through an approximation of the succession of tasks causing response time, called Longest-Latency Path.

— Chapter 6 - Contribution 3: An evaluation of an MoA-based approach for estimating system execution latency of functionalities characterized by LLPs as application activity is proposed. Although, the method is far from providing a complete solution in terms of generalization to mappings, application parameters, application and architectures, it represents the first attempt of building an elastic model aimed at supporting runtime management in the intricate context of MPSoC timing analysis.

Future Works

Many works can be envisioned to enhance the contributions of this thesis. In order to reduce limitations and complete the design steps to reach system adaptation, some propositions of future work are exposed below:

— Chapter 4 - Contribution 1: The DFF HLS is based on the combination of the HLS for streaming functionalities named CAPH and the suite for CGR devices named MDC. This contribution has required a large effort of tool design within the context of the H2020 CERBERO project. Nevertheless, the degree of design expressivity of the CAPH-MDC combination is currently limited compared to that of the single tools. In particular, restrictions to the potential of the proposed flow are related to the generation of the CGR accelerator, since the specifications provided by CAPH are not fully supported by MDC. In addition, the integration of the accelerator in design solutions obtained through the proposed adaptation framework is still manually managed. Indeed, using the drivers provided by the generation of the accelerator, the developer needs to manually create a SW task in order to properly provide and obtain input and output data respectively to/from the CGR PE, which is reconfigured by means of a specific parameter given as an input to the drivers of

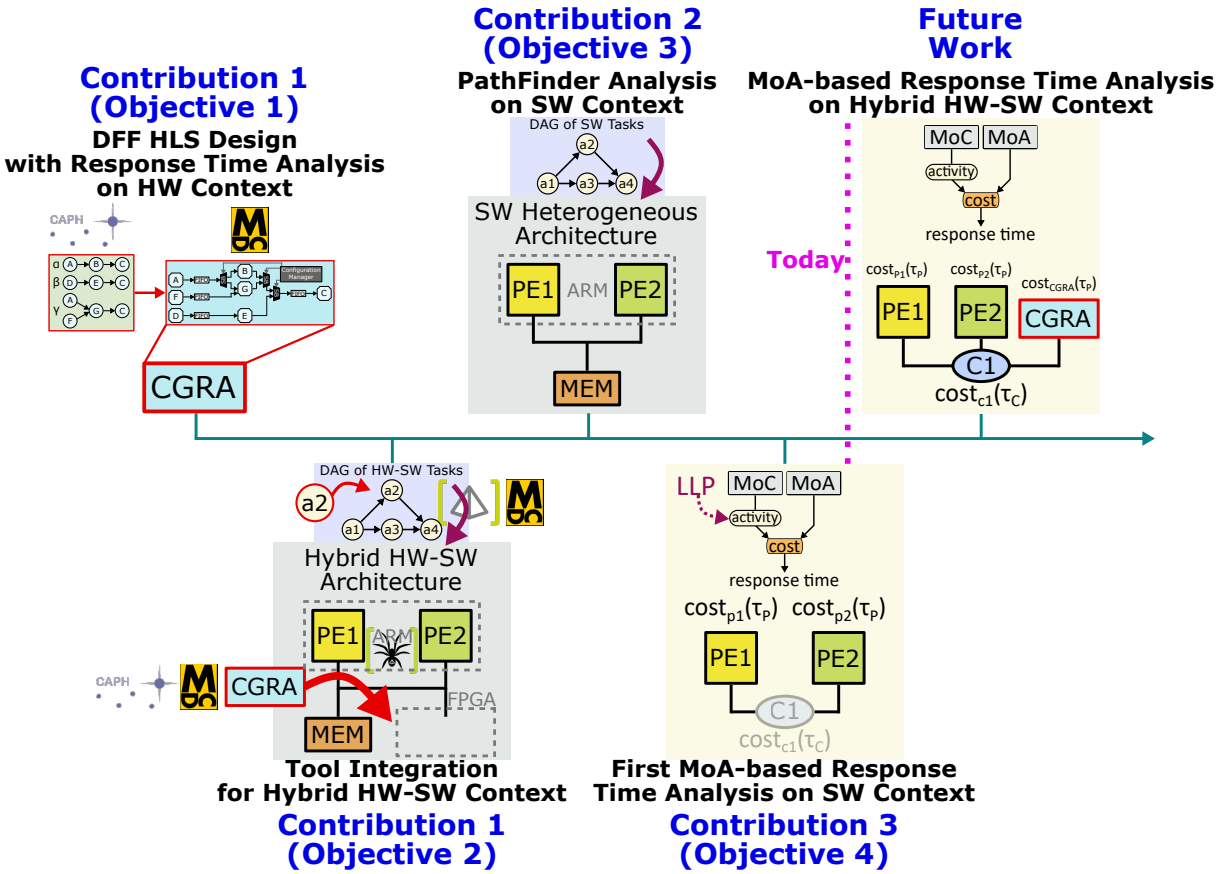


Figure 7.1 – Development timeline of the thesis work according to the contributions and objectives presented in Section 1.

the accelerator. Moreover, from a high-level perspective, the accelerator appears into the System-Level Architecture Model as a generic SW PE, without the attribution of specific characteristics, such as the property of the number of configurations present in the IP, and an extended architecture model is to be created to improve integration. The same applies to the configuration parameter described in the Parameterized and Interfaced SDF, which should actually be limited to a limited set of values, while currently they can take any integer value. This limitation is also linked to the characterization of each configuration of the accelerator in terms of response time in order to properly schedule the functionality on the target MPSoC. On the other hand, the switching of the parameters configuration is applied to the subsequent firing of the whole application graph PiSDF. This strategy may be unsuitable in the context in which the timing overhead of reconfiguration is severely constrained. Finally, further demands may derive from the exploitation of parallel execution of

multiple accelerators, since only one CGR IP is tested in the proposed PoCs.

— Chapter 5 - Contribution 2: PathFinder provides a set of execution time contributions total to the response time which include interferences in reference to a selected critical path. Limitations related to the types of interference not considered in the LLP analysis can affect the estimation in terms of accuracy. In fact, as already mentioned, high-order combinations of the so-called *scheduling* and *dependency* interferences can arise in certain use cases, although the amount of processing dominates the communication and synchronization between tasks. For these reasons, further studies aimed at extending the applicability of the LLP analysis would be interesting. In addition, the algorithmic complexity of the PathFinder can be reduced through refinements of the choices associated with the DAG exploration and CP detection. Finally, a design-time strategy aimed at obtaining the LLP is highly desirable, since this is currently based on a-posteriori timing information (level E in Section 3.2.2). LLP generalization to sets of system configurations is especially desirable if using the LLP as the activity for an MoA-based response time study.

— Chapter 6 - Contribution 3: As a first proposition with respect to a highly complex context of the timing analysis of parametric functionality mapped on a heterogeneous MPSoC, the estimation of the response time based on MoAs represents an on-going work. In particular, a further study on a generalized activity obtained at design time, also based on the LLP criteria, is required. For instance, a strategy aimed at exploiting a single LLP for more than one mapping could be explored. This could be applied first to span-dominated applications, which present a limited interference with respect to the CP. On the other hand, a method for predicting the LLPs of parametric functionalities can represent another way to obtain activity. In addition, new MoAs described as non-linear models, differently from the chosen LSLA, may favor accuracy and fidelity, but they require to stick to explainable parameters for the activity and MoA to remain explainable. As a further work, the assessment of functionalities with HW task mapped on CGR accelerators can be considered. For this purpose, a fundamental step consists of integrating the MoA-based estimation into the proposed adaptation framework, in order to handle the parametric functionality upon heterogeneous systems.

Appendix A

French Summary

A.1 Introduction

Actuellement, les systèmes embarqués sont certainement les dispositifs les plus courants dans l'électronique de masse et industrielle. Les systèmes multiprocesseurs sur une seule puce (MPSoCs), sur lesquels sont traitées leurs phases de traitement, intègrent un nombre croissant de éléments de traitement (PEs) hétérogènes afin d'assurer efficacement les *fonctionnalités* modernes. Pour cette raison, la conception de systèmes basés sur MPSoCs est devenue de plus en plus complexe et plusieurs approches visant à limiter cette complexité sont apparues [CF16]. Parmi les concepts de systèmes embarqués, les systèmes cyberphysiques (CPSs) ont été connus et étudiés avec intérêt par la communauté scientifique au cours des dernières années. Ces systèmes sont capables de surveiller et de contrôler des éléments physiques et considèrent des composants hétérogènes qui interagissent les uns avec les autres selon des modalités différentes en fonction du contexte dans lequel ils fonctionnent [Raj+10]. La conception et la maintenance de ces systèmes sont extrêmement complexes en raison de leur nature multidisciplinaire, de leurs exigences complexes, de l'hétérogénéité de leurs composants et de la communication continue entre leurs couches physiques et cybernétiques [DLS12]. Tant les systèmes embarqués que les CPSs impliquent des exigences élevées en termes de *flexibilité* et d'efficacité. La première propriété exige des solutions capables de réaliser différentes *fonctionnalités* évaluées dans divers *modes de fonctionnement*. La seconde est nécessaire pour l'exploitation du système en ce qui concerne les performances et les coûts découlant des choix de conception (par exemple en ce qui concerne les exigences liées au budget énergétique et au *temps de réponse*).

Afin d'atteindre la *flexibilité* et l'efficacité, il est nécessaire de disposer de matérielles et logicielles numériques capables de mettre en œuvre différentes *fonctionnalités* avec plusieurs *modes de fonctionnement* afin de répondre aux exigences. Une solution de plus en plus envisagée est représentée par la combinaison d'implémentations reconfigurables

à la fois en termes de matérielles et logicielles [Sha+14 ; GPK19 ; Pér+20]. Toutefois, en présence de multiples *modes de fonctionnement* qui dépendent de l'état du système et de l'environnement extérieur, la *reconfiguration* peut ne pas être suffisante pour garantir une *flexibilité* complète et/ou un degré d'efficacité approprié. En effet, dans des cas tels que la défaillance de composants du système ou la nécessité de changer de fonctionnalité, une gestion de l'exécution en temps réel est nécessaire. Néanmoins, l'adaptation en fonction d'événements incertains et de l'évolution des exigences fonctionnelles et non fonctionnelles constitue un défi important pour les développeurs de systèmes. Afin de faciliter la *reconfiguration* et l'application des stratégies de *adaptation*, l'utilisation de paramètres associés à des *modes de fonctionnement* spécifiques offre une gestion de haut niveau de la *fonctionnalité*. Étant donné que ces *modes de fonctionnement* peuvent concerner à la fois les implémentations matérielles et logicielles, les *fonctionnalités* paramétrique réalisée sur des systèmes hétérogènes, visant à apporter *flexibilité* et efficacité, est de plus en plus évaluée.

Parmi les indicateurs clés de performance (KPIs) de systèmes embarqués et CPSs les plus courants, le *temps de réponse* joue un rôle pertinent pour les stratégies envisagées pour la *reconfiguration* et *adaptation* des solutions de traitement du signal [HAR14 ; GPM14]. En effet, cette métrique représente le temps d'exécution d'une élaboration complète d'une *fonctionnalité*, à partir de l'acquisition d'une donnée d'entrée donnée et jusqu'à la génération d'une donnée de sortie associée. Néanmoins, l'évaluation précise du *temps de réponse* implique des analyses de temps complexes et/ou de fortes limitations dans la mise en œuvre des composantes matérielles et logicielles. En effet, bien que différentes approches dans la littérature visent à fournir une précision et une fiabilité dans la vérification du *temps de réponse*, leur applicabilité dépend de divers aspects, tels que : la disponibilité des informations concernant les détails matérielles et logicielles, et la quantité de traitement et de demandes dans l'utilisation des ressources du système associées à la *fonctionnalité* [Wil+08]. En outre, dans les contextes exigeant une *reconfiguration* et une *adaptation*, l'évaluation du calendrier et la budgétisation en ligne peuvent nécessiter l'exécution de modèles complexes, non compatibles avec les exigences de performance applicative. Par conséquent, ce problème est loin d'être résolu efficacement pour les *fonctionnalités* qui exigent *flexibilité* et efficacité dans leur exécution. C'est pourquoi il est nécessaire d'identifier une autre approche afin de simplifier l'estimation du temps de réponse, en particulier lorsque les *fonctionnalités* paramétriques sont exécutées sur des systèmes hétérogènes. Toutefois, la réduction de la complexité n'est pas gratuite et est généralement liée à une

perte de précision de l'évaluation *temps de réponse*. L'analyse doit donc maintenir la robustesse des valeurs estimées, en présentant au moins un degré élevé de fidélité par rapport à la tendance du coût réel de la *fonctionnalité* en termes de *temps de réponse*.

Le résumé expose tout d'abord un aperçu de l'état de l'art relatif à l'approche proposée (Section A.2). La Section A.3 présente la définition du problème, la motivation et les objectifs de chaque contribution. La Section A.4 décrit les contributions de la thèse comme suit. La Section A.4.1 (Contribution 1) présente et évalue un flux de conception capable d'estimer le *temps de réponse* dans la conception des accélérateurs CGR (Objectif 1) et un cadre pour le développement d'applications paramétriques mappées sur PE reconfigurable à gros grains (CGR) et d'usage général (objectif 2). La Section A.4.2 (Contribution 2) propose une méthodologie pour détecter les facteurs déterminants du *temps de réponse* dans l'exécution des *fonctionnalités* paramétriques mappées sur des MPSoCs hétérogènes (Objectif 3). La Section A.4.3 (Contribution 3) décrit un flux de conception pour l'estimation de la *latence d'exécution du système* par la MoA linéaire, en fonction de l'activité définie dans la contribution 2 (objectif 4). Les considérations finales relatives au travail de thèse et les éventuelles améliorations futures concluent l'étude (Section A.5).

A.2 Etat de l'Art

Comme ce travail de thèse est axé sur l'estimation du *temps de réponse* des *fonctionnalités* paramétriques réalisées sur des systèmes composés de systèmes polyvalents et de CGR PE, cette section donne un aperçu des domaines de recherche liés à ce contexte. En effet, l'évaluation de ces KPI dépend des stratégies utilisées dans la conception de la *fonctionnalité*. En particulier, une telle conception peut combiner des approches spécifiques associées au développement de mises en œuvre matérielles et logicielles de la *fonctionnalité*. Toutefois, selon le modèle d'application et la plate-forme cible considérés dans la conception, l'analyse du temps varie considérablement en termes de complexité et de précision. En fait, le comportement d'une application peut être décrit par une combinaison d'éléments opérationnels présents dans un ensemble spécifique correspondant à un ensemble appelé modèle de calcul ou Model of Computation (MoC) [LX04]. Un MoC vise à formaliser les représentations des applications et implique une stratégie spécifique d'évaluation du *temps de réponse*. D'autre part, l'impact de l'architecture cible sur le temps de réponse est communément connu, car celle-ci peut avoir différents types et nombres de composants matériels. En général, comme la description des *fonctionnalités* par des

spécifications basées sur les MoCs DF présente des propriétés telles que l'abstraction et la modularité, elle a été largement utilisée dans l'évaluation du *temps de réponse*. C'est pourquoi une attention particulière sera accordée à ces MoCs et à l'utilisation d'architectures composées de PEs polyvalents et spécifiques.

A.2.1 Conception d'applications de traitement du signal

Nous appelons application de traitement du signal une *fonctionnalité* de traitement numérique à effectuer sur des données numériques arrivant constamment. Le modèle utilisé pour décrire une telle *fonctionnalité* détermine la stratégie d'estimation du *temps de réponse* [AS19 ; Men+17 ; Objb ; ABA16]. Différentes stratégies de Design Space Exploration (DSE) sont basées sur des MoCs spécifiques pour étudier les KPIs d'application [AA18]. Cependant, les méthodes DSE peuvent être regroupées dans les quatre familles ci-dessous, sur la base de leurs modèles d'application.

Simulation de code impératif (Imp). La *latence* d'un code impératif avec des tâches parallèles peut être prédit avant l'exécution réelle en utilisant un Instruction Set Simulator (ISS) [AS19]. Cette stratégie nécessite une connaissance détaillée de l'architecture sous-jacente et souffre également d'un manque d'évolutivité, car les opérations matérielles rapides sont simulées par un logiciel qui est d'un ordre de grandeur plus lent [Pim17].

SystemC-based simulation (SysC). Les bibliothèques C++ de SystemC [Acc] fournissent des mécanismes d'annotation du temps pour spécifier de manière modulaire un algorithme et simuler son comportement temporel avec plusieurs niveaux de précision. Toutefois, la disponibilité limitée de modèles de plates-formes polyvalentes sur lesquelles simuler les applications rend l'utilisation de la simulation du Système C assez coûteuse en termes de temps de conception dans la pratique.

Simulation basée sur UML (UML). Le modèle d'objet standard Unified Modeling Language (UML) décompose un calcul en un ensemble d'objets. Les extensions UML telles que SysML [Obja] et MARTE [Objb] fournissent une sémantique de spécification des KPIs pour alimenter une simulation abstraite. Dans ces outils DSE, l'estimation de la latence est effectuée par des simulations RTL ou TLM. Cependant, il n'existe pas de procédure standard pour dériver les KPIs pour un ensemble spécifique de ressources matérielles et de caractéristiques non fonctionnelles.

Évaluations des KPIs basées sur un modèle de flux de données (DF). Lors de l'utilisation d'un MoC DF, un des nombreux modèles abstraits alternatifs peut être

choisi pour représenter le calcul d'une *fonctionnalité* comme un ensemble de tâches non préemptives qui échangent des messages entre elles par le biais de FIFOs [PL95]. Les évaluations de latence peuvent être calculées à partir d'une instance du MoC DF et d'un modèle de l'architecture matérielle [Mat+19], et la précision de l'analyse de la latence dépend de la qualité de ces modèles de latence.

Dans cette thèse, une méthode appelée PathFinder, visant à estimer *latence* dans le contexte DF, est proposée. Cette approche présente des capacités de modularité et d'abstraction qui ont été largement exploitées dans l'analyse du temps. En outre, cela s'adapte naturellement à la prévisibilité de la *latence* dans des architectures hétérogènes. En littérature, une pléthore de MoCs DF a été défini. Leur sémantique représente une modélisation différente en termes d'activation, d'exécution et d'interaction entre les tâches qui décrivent la *fonctionnalité* souhaitée. Dans cette thèse, le MoC PiSDF a été choisi pour la disponibilité d'outils avancés et la prévisibilité des représentations d'applications supportant la *reconfiguration*.

A.2.2 Conception d'architectures reconfigurables à gros grains

Au cours des dernières décennies, des systèmes composés de différentes plateformes ont été conçus pour optimiser la *flexibilité* et l'efficacité. En effet, les architectures d'usage général permettent des solutions plus flexibles avec un effort de conception et un coût moindre par rapport aux architectures reconfigurables ou ASICs. Néanmoins, l'*accélération* matérielle est parfois nécessaire pour les charges de travail à forte intensité de calcul afin de satisfaire aux exigences du cas d'utilisation [FW19], notamment en termes de timing et de consommation d'énergie/puissance. Comme cette thèse se concentre sur le premier aspect, et en particulier sur l'estimation de la *latence* pour les systèmes flexibles, une architecture hétérogène basée sur une combinaison d'accélérateurs CPUs et de CGR a été sélectionnée. En fait, il s'agit d'un compromis entre la *flexibilité* et l'efficacité de la co-conception matérielle/logicielle, qui permet d'éviter des frais de conception élevés pour gagner en performance. Les dispositifs reconfigurables consistent en un ensemble de PEs et de connexions qui peuvent être adaptées à l'exécution [CH02], afin de réaliser une *fonctionnalité* ou un *mode de fonctionnement* spécifique (appelé aussi *scénario*). L'*adaptation* du système est appliquée par une *reconfiguration* au niveau matériel des blocs logiques et des interconnexions [TB01]. Ceux-ci sont déployés sur un substrat logique d'une plate-forme reconfigurable, dont le plus courant sur le marché est l'FPGA [Res]. Dans la littérature, cette plate-forme est utilisée pour mettre en œuvre deux ap-

proches principales basées respectivement sur les architectures reconfigurables FG et CG [Che05]. La *reconfiguration* FG consiste à remplacer totalement ou partiellement la description de l'architecture au niveau des bits dans l'FPGA. En revanche, la *reconfiguration* dans l'approche CG est une sélection des PEs ou de leur configuration étant donné le *mode de fonctionnement* représenté comme un ID et choisi pour l'exécution de la *fonctionnalité*. Bien que les systèmes CGR permettent moins de *flexibilité* par rapport aux homologues FG, ils évitent les temps de *reconfiguration* lents requis par ces derniers. Pour ces raisons, au fil des ans, plusieurs CGRAs ont été proposés, qui peuvent être regroupés dans les principales catégories introduites dans [Har01]. Cette classification se concentre sur les différences en termes de disposition et de connexions des PEs, en distinguant trois types différents : les tableaux linéaires, les tableaux à base de maillage et les tableaux à base de barres transversales. En plus des catégories mentionnées ci-dessus et proposées dans [Har01], les systèmes CGR peuvent intégrer des ensembles de PEs hétérogènes (par ex. DSPs, processeurs, etc.) et FUs (par ex. additionneurs, multiplicateurs, etc.) et les interconnexions strictement nécessaires à la *fonctionnalité* spécifique au domaine.

A.2.3 Analyse de temps sur les MPSoCs

En se concentrant sur l'analyse temporelle des systèmes à plusieurs PEs, l'enquête présentée dans [Mai+19] classe l'état de l'art en quatre catégories générales de recherche, comme suit.

Full Integration (FI). Fonctionne sur la base d'un effet de levier FI sur l'information complète concernant l'exécution des tâches. L'analyse est basée sur WCET, en utilisant la vérification du modèle et les états abstraits du processeur. Bien que FI promette des estimations précises, cette stratégie souffre généralement d'une grande complexité et d'une faible scalabilité.

Temporal Isolation (TI). TI sépare le traitement des tâches dans le temps afin d'éviter l'utilisation simultanée des ressources partagées, principalement le bus mémoire, qui est considéré comme la principale source d'*interférence*. TI conduit à une évaluation du temps plus simple que les analyses FI. Cependant, la moindre utilisation du matériel pénalise les performances de l'application dans son ensemble.

Integration into Schedulability Analysis (IS). IS compte des retards supplémentaires dus à la présence de tâches en co-exécution et à la politique d'accès aux ressources partagées dans l'analyse de planifiabilité. Les travaux basés sur l'approche IS gèrent efficacement l'analyse avec un accès sérialisé aux ressources partagées, au prix de l'exclusion

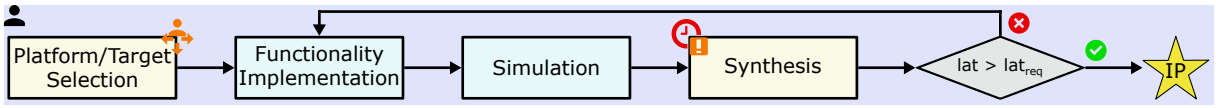
de leur utilisation superposée.

Mapping and Scheduling (MS) MS combine une analyse du temps avec des solutions de mapping et de planification pour garantir les contraintes de temps. Les travaux MS souffrent de problèmes d'évolutivité, ainsi que d'une forte dépendance aux caractéristiques de l'application et de l'architecture. Pour ces raisons, elles ont tendance à ne pas fournir une solution complète pour l'ensemble du problème auquel leur approche est confrontée.

A.3 Définition du Problème et Motivation

A.3.1 Définition du problème dans le contexte du matériel du CPS

La connaissance de la *latence* du système mis en œuvre dès les premières étapes de la conception est un outil important pour améliorer la productivité et réduire l'effort de conception. La Figure A.1A illustre la méthodologie de conception des principaux



(A) Approche dépendant de la plate-forme/du cible avec informations post-synthèse

(B) Approche agnostique des plates-formes/cibles avec informations de pré-synthèse

FIGURE A.1 – HW time verification strategies.

outils HLS à base d'impératifs et dépendant de la cible présents à l'état de l'art. Ceux-ci nécessitent une DSE pour obtenir la *latence d'exécution du système* requise ($lat > lat_{req}$), en itération entre l'implémentation, la simulation et la synthèse. De plus, afin de déployer un IP correspondant pour une nouvelle cible hardware, la *fonctionnalité* doit être réimplémentée selon les spécifications associées à la nouvelle cible. Pour ces raisons, les développeurs doivent avoir une connaissance approfondie du synthétiseur et un effort considérable est nécessaire pour obtenir une architecture optimisée, en particulier dans les contextes nécessitant des cibles différentes. La Figure A.1B illustre le DFF HLS proposé

dans cette thèse, qui vise à offrir une évaluation *du temps de réponse avant la synthèse*, en s'appuyant sur des outils basés sur les flux de données et la programmation fonctionnelle. Cela donne aux développeurs l'avantage de se concentrer sur l'application à accélérer avec une approche orientée vers les *fonctionnalités* plutôt qu'une approche basée sur la synthèse. De plus, lorsque la description de l'algorithme conduit à un HDL synthétisable à l'aide d'une plate-forme/cible, le développement de l'IP peut être divisé en deux parties associées respectivement à l'application et au matériel.

A.3.2 Définition du problème dans le contexte du logiciel parallèle du CPS

Quel que soit le type de charge de travail applicatif, le *temps de réponse* est déterminé par une succession de mécanismes causaux, longs à mettre en œuvre, qui sont généralement représentables par un DAG de tâches dépendantes des données, représentant une itération d'exécution [FW19]. Au niveau matériel, les MPSoCs hétérogènes sont des solutions efficaces pour l'exécution d'applications multifonctionnelles. Cependant, les performances du système en termes de *temps de réponse* sont difficiles à prévoir et à comprendre à partir des modèles d'application et d'architecture. En effet, *latence* est une propriété hautement non linéaire, affectée par de nombreux phénomènes logiciels, matériels et d'ordonnancement. Cette thèse présente une nouvelle méthodologie de modélisation de la *latence d'exécution du système*, nommée PathFinder, avec pour objectif d'extraire, à partir d'un modèle d'application, un Longest-Latency Path (LLP), c'est-à-dire un sous-ensemble de la charge de travail de l'application qui se rapproche du *temps de réponse*. En généralisant ce concept, nous appelons *activité* la part de la charge de travail d'une application qui détermine un KPI donné [Pel+18].

A.3.3 Définition du problème dans le contexte du matériel/logiciel hybride

Les sections A.3.1 et A.3.2 ont introduit les objectifs visant à estimer respectivement le *temps de réponse* des *fonctionnalités* implémentées comme un accélérateur CGR, et le DAG des tâches logicielles. Bien que ces deux aspects aient été traités séparément, leur combinaison est souvent souhaitée dans les contextes où la *reconfiguration* ou l'*adaptation* est nécessaire. Cependant, la planification et le mappage de ces *fonctionnalités* nécessite de généraliser la stratégie utilisée pour estimer la *latence d'exécution du système*.

Afin de déterminer le *temps de réponse* d'une *fonctionnalité* paramétrique, il faut envisager de nouvelles approches qui font un compromis entre précision et complexité. En particulier, l'idée est d'évaluer une stratégie qui favorise une réduction de la complexité au détriment d'une dégradation contrôlée de la précision. Cette nouvelle approche doit néanmoins générer un modèle de haute fidélité, c'est-à-dire une capacité à alimenter des décisions correctes. La fidélité peut être définie par les coefficients de corrélation du rang de Spearman ρ et du tau de Kendall τ [BB00 ; JIP10].

Récemment, une nouvelle approche de modélisation de l'architecture appelée MoA et basée sur les concepts d'abstraction et de modularité a été proposée [Pel+18]. Dans le cas de l'estimation de l'énergie, le degré de fidélité a été démontré très prometteur (FM_τ égal à 0,86 et 0,93 pour les systèmes hétérogènes basés sur CPUs et GPUs respectivement). Néanmoins, les études sur les MoAs [Pel+18 ; Pay+19] sont axés sur l'énergie, qui, à la différence de *la latence d'exécution du système*, présente des propriétés additives par nature.

A.4 Contributions

A.4.1 Construction d'une chaîne d'outils de conception pour des CPS matérielles et logicielles flexibles et prévisibles

Le flux proposé [Rub+19] s'appuie sur deux outils préexistants : le compilateur CAPH [1] [SBA13] et le jeu d'outils MDC [2] [Pal+17]. Sur la base des compléments décrits, ce travail propose une chaîne d'outils entièrement automatisée pour la spécification et le déploiement des DSAs CGR. Le flux DFF HLS proposé se compose de trois phases principales (voir Figure A.2) : *Composition*, qui effectue une compilation modèle-à-modèle et produit une DFN multi-dataflow de haut niveau du DSA ; *Optimisation*, pour le dimensionnement optimal des FIFOs en relation avec les acteurs ; *Generation*, le déploiement du DSA CGR. En résumé, en ce qui concerne le contexte des DSAs CGR, les principaux avantages/caractéristiques du flux intégré proposé sont les suivants : *Custom PE Generation*, génération de PEs hétérogènes pour chaque acteur du flux de données ; *Reconfigurability Management*, mécanisme basé sur le flux de données qui maximise et contrôle la réutilisation des ressources ; *Predictability*, avant la synthèse, des estimations de *latence* peuvent

1. `https://github.com/jserot/caph`
2. `https://github.com/mdc-suite`

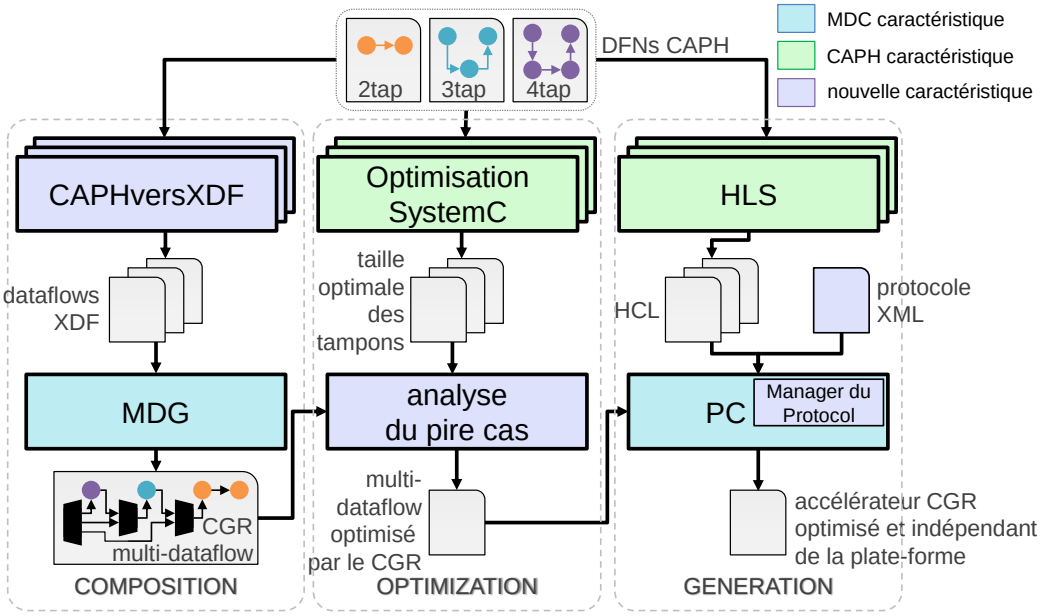


FIGURE A.2 – DFF HLS proposé.

être effectuées sur la base des DFNs CAPH prétraitées ; *Target Independence and Availability*, MDC et CAPH sont agnostiques à la plate-forme et open source ; *Code Readability*, MDC et CAPH préservent la correspondance entre les acteurs du flux de données et les PEs matériels. Le DFF HLS proposé a été évalué par rapport aux principaux outils HLS commerciaux (Vivado HLS v2015.2 et Intel FPGA SDK for OpenCL v17.0) dans l'implémentation reconfigurable de filtres approximatifs HEVC (1D et 2D) [Nog16]. Les résultats suggèrent que DFF HLS est une alternative prometteuse aux HLS impératifs classiques pour construire des DSA CGR flexibles et prévisibles, puisqu'il fournit une utilisation compétitive des ressources et des prévisions de latence pré-synthèse.

Afin d'exploiter l'accélération CGR au niveau système des architectures multi-coeurs, l'activité d'intégration entre le gestionnaire d'exécution basé sur le flux de données SPIDER [3] [Heu+14] et la suite de flux de données vers le matériel MDC a été menée. En effet, leurs propriétés complémentaires ont permis une nouvelle approche basée sur la modularité du flux de données et la gestion de l'exécution de la *reconfiguration* matérielle (Figure A.3). Cette intégration a permis la création des preuves de concept présentés qui ont été utilisés dans l'évaluation de travaux ultérieurs dans le cadre du projet CERBERO H2020.

3. `https://github.com/preesm/spider`

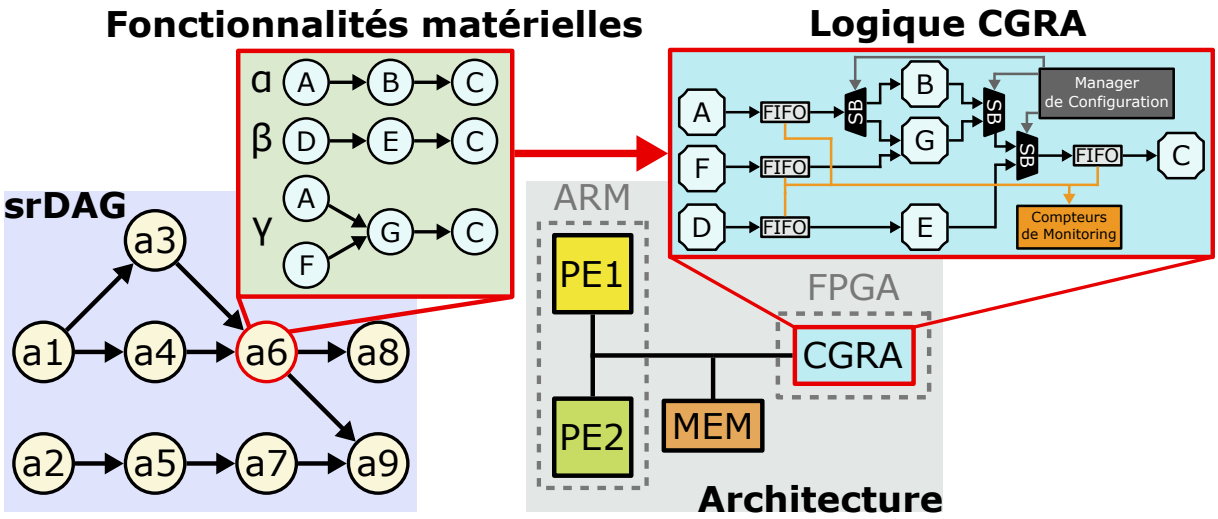


FIGURE A.3 – Vue d'ensemble du système hétérogène reconfigurable par matériel/logiciel.

A.4.2 PathFinder : étude de l'activité à l'origine de la latence de l'application

La méthode PathFinder calcule l'activité de l'application, sous la forme d'un Longest-Latency Path (LLP), en déterminant le *temps de réponse* des applications modélisées par de tels DAGs de tâches, marqués avec des temps pour l'exécution de la tâche et le passage du message. Dans les résultats expérimentaux, ces DAGs de tâches sont générés à partir d'applications modélisées par des flux de données utilisant le MoC PiSDF [Des+13]. La définition choisie du temps de réponse correspond à la valeur W dans la formule de mise en file d'attente [Lit61]. La valeur W donne *le temps prévu passé par une unité [de données] dans le système* lorsqu'elle passe d'une source de données à un puits de données. Afin de rendre l'étude plus pratique, plusieurs sources et puits de données peuvent être considérés, déterminés par le concepteur du système et basés sur les objectifs du système. Le *temps de réponse* dérivé est le maximum de ces latences indépendantes. Dans la mesure de nos connaissances, ce travail constitue le premier effort pour décomposer les facteurs déterminants de la *latence d'exécution du système* dans un MPSoC. Un LLP est un ensemble de contributions de *latence* provenant d'une chaîne source-puit du DAG, le Critical Path (CP), avec l'ajout des éléments du DAG qui interfèrent avec l'exécution idéale de le CP. Le flux de conception pour l'estimation de la *latence* LLP pour une application paramétrique décrite comme un graphique de tâches est proposé. Comme le montre la Figure A.4, cela commence par la création de l'application et la modélisation de l'architecture

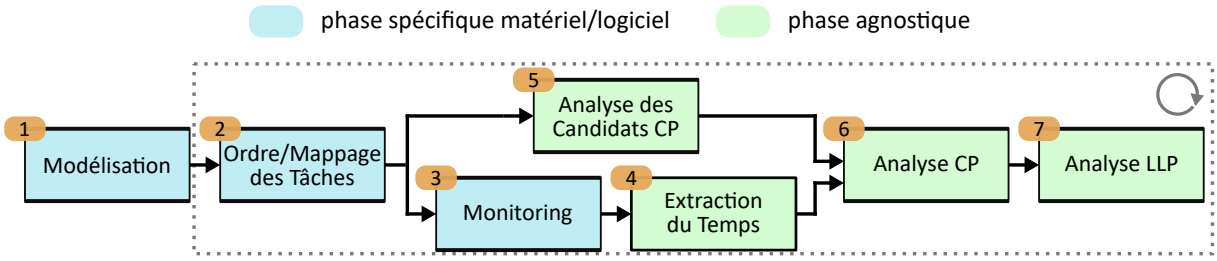


FIGURE A.4 – Flux de conception mettant en oeuvre la méthode PathFinder.

cible (1). Les phases suivantes peuvent être réalisées en boucle pour chaque *scénario*, à partir de l'ordonnancement (2). L'exécution de l'application est surveillée (3). Ainsi, en fonction de la métrique choisie, le coût du timing de l'acteur et des instances de bord sont extraits des mesures (4). En parallèle, l'analyse des candidats LLP peut être évaluée (5). Selon cette méthode, le CP est obtenu à partir des temps d'acteurs surveillés pour une exécution souhaitée (6). Enfin, l'analyse LLP est effectuée (7).

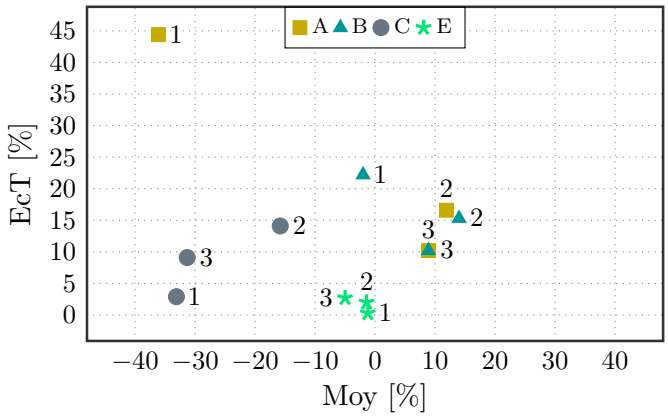


FIGURE A.5 – Comparaison des erreurs par rapport au niveau de référence D. Pour chaque niveau, le pourcentage d'erreur en termes d'écart type (EcT) et de moyenne (Moy) le long des *scénarios* est rapporté pour les 3 applications : Stabilisation vidéo (1), Stéréo (2) et Calcul du point SIFT (3).

Le flux décrit dans la Figure A.4 a été évalué avec 3 applications PiSDF différentes évaluées dans plusieurs *scénarios* et mappées sur une carte ODROID-XU3 [Har]. PiSDF. [Des+13] est un MoC DF qui modélise la synchronisation entre les tâches et la sémantique algorithmique traitée en utilisant des paramètres de haut niveau. Chaque configuration des paramètres PiSDF représente un *scénario* spécifique de l'application, ayant un certain nombre d'instances d'acteurs. La Figure A.5 propose une comparaison entre 4 différents niveaux d'information sur la plate-forme matérielle et l'exécution surveillée (niveau D).

Ces 4 niveaux sont les suivants : aucune information sur l'architecture n'est prise en compte et le CP est considéré indépendamment sur le système (niveau A) ; connaissance de un nombre connu de PEs homogènes et mémoire limitée (niveau B) ; distinction des types de PEs dans le système cible (niveau C) ; et PathFinder (niveau E).

Pour évaluer le degré de similitude entre deux diagrammes de Gantt sur l'exécution des tâches X et Y, le concept de *Jaccard index* [Rez+19], aussi appelé Intersection over Union (IoU), a été utilisé. Une nouvelle métrique appelée Gantt Similarity (GS) qui donne un score entre 0 (aucune correspondance) et 1 (correspondance complète) a été définie :

$$GS(X,Y) = \sum_{i=0}^{T} [IoU(X_i, Y_i)] \cdot w_i = \sum_{i=0}^{T} \left[\frac{area(X_i \cap Y_i)}{area(X_i \cup Y_i)} \right] \cdot \frac{area(Y_i)}{area(Y)} \tag{A.1}$$

où T est le nombre d'acteurs présents dans l'srDAG, et X_i et Y_i représentent les cases associées aux temps d'exécution de la tâche i-th dans respectivement X et Y, et w_i prend en compte la façon dont les tâches affectent l'exécution avec les différents poids.

L'évaluation de la méthode sur les 3 applications montre les difficultés à prévoir et à comprendre le *temps de réponse* en fonction des informations a priori. D'autre part, PathFinder fournit des estimations de *latence* de haute précision et une connaissance de l'exécution basée sur les applications décrites comme un srDAG ainsi que sur les temps d'exécution surveillés.

A.4.3 Estimation du temps de réponse par l'activité LLP et une MoA

La combinaison de la notion de LLP et d'un MoA linéaire est une direction prometteuse pour décomposer le problème de la généralisation du modèle du *temps de réponse*. Lorsqu'une activité est mappée sur un MoA, le coût dû à son exécution peut être estimé, soit au moment de la conception si tout est connu sur cette activité, soit au moment de l'exécution si l'activité est *découverte* en cours d'exécution. Le MoA considéré dans cette thèse est le Linear System-Level Architecture Model (LSLA) proposé dans [Pel+18], qui correspond à un graphique non orienté ($\Lambda = (P, C, L, cost, \lambda)$) respectivement composés d'ensembles d'éléments de traitement PEs, de nœuds de communication CNs, de liens de connexion, de fonctions de coût et de coefficients du rapport de coût entre traitement et communication. Une contrainte pour que le modèle soit un MoA est que, pour être reproductible, il doit spécifier le calcul des coûts. La quantité de "pression" exercée sur le matériel par l'appli-

cation est abstraite par une notion de *activité* composée de *tokens*. Lorsqu'un jeton τ est mappé sur un noeud n (a PE ou a CN), son coût en LSLA est calculé par une fonction définie comme suit : $cost_n = \alpha_n \cdot size(\tau_n) + \beta_n = \alpha_n s_n + \beta_n$, où α_n et $\beta_n \in \mathbb{R}$. La différence entre l'énergie et le *temps de réponse* réside dans le calcul du coût du modèle. En effet, le coût de l'exécution de l'activité basée sur LLP A sur le MoA LSLA Λ avec $size(P)$ PEs et $size(C)$ CNs peut être estimé comme suit : $cost(A, \Lambda) = [\sum_{\tau_p}^{T_p \subset LLP} cost_p] + \lambda \cdot [\sum_{\tau_c}^{T_c \subset LLP} cost_c]$, avec $cost_p = \alpha_p s_p + \beta_p$ et $cost_c = \alpha_c s_c + \beta_c$, et où $p \leq size(P)$ ($c \leq size(C)$), avec $size(P)$ ($size(C)$) le nombre de PEs (CNs) présent dans le système. p correspond à l'index d'un PE spécifique (CN) sur lesquels les jetons sont mis en correspondance. Ceux-ci représentent l'activité en tant que contributions appartenant au LLP. Dans les contextes où l'activité applicative est principalement due au traitement, qui domine la communication, le coût de l'exécution devient : $cost(A, \Lambda) \approx \sum_{\tau_p}^{T_p \subset LLP} cost_p$.

Le flux de conception utilisé pour former et évaluer les LSLA MoA avec l'activité basée sur LLP part de la sélection de i) s *scénarios* d'application, décrits à travers les spécifications données par le MoC DF avec un ensemble déterminé de paramètres, et ii) l'architecture à modéliser, qui intègre K PEs configurés pour avoir k d'eux actifs. Parmi tous les mappages possibles, M solutions souhaitées sont choisies. Celles-ci sont analysées par PathFinder afin d'obtenir autant de LLPs. Dans ce flux, l'LLPs représente l'activité d'un seul srDAG lié à un *scénario* spécifique. Par conséquent, un certain nombre m de LLPs est utilisé dans l'apprentissage des fonctions de coût. En sortie de cette phase, des ensembles m de paires (α,β) associés aux types t de PEs présents dans le modèle d'architecture sont fournis. À ce stade, les fonctions de coût complètent l'ensemble du modèle. Ce modèle est prêt à être évalué avec les autres $m' = M - m$ LLPs afin d'évaluer sa précision et sa fidélité [BB00 ; JIP10] dans un cas de mappage généralisée et sous l'hypothèse que l'LLP est bien connu, en utilisant les équations suivantes : i) $FM_\rho = 1 - \frac{2 \cdot \sum_{i=1}^{n} r_i^2}{\frac{n(n^2-1)}{3}} = 1 - \frac{2 \cdot \sum_{i=1}^{n} (P_i^{r,a} - P_i^{r,e})^2}{\frac{n(n^2-1)}{3}}$, et ii) $FM_\tau = \frac{n_c - n_d}{\frac{1}{2} n(n-1)}$.

Le cas d'utilisation choisi pour l'évaluation du model correspond à $M = 24$ différentes solutions de mappage d'un *scénario* de l'application Stereo Matching exécutée sur une carte ODROID-XU3 [Har], qui est composée de $K = 8$ de cœurs ARM (4 CORTEX-A7 et 4 CORTEX-A15, $t = 2$). Cette première évaluation montre la possibilité d'obtenir une bonne fidélité ($FM_\rho \geq 0.94$, $FM_\tau \geq 0.83$). L'LLP peut être considéré comme utile dans la vérification de l'exactitude et de la fidélité d'un MoA basé sur cette activité, mais une question importante qui reste ouverte est de savoir comment généraliser également l'LLP aux cas non vus.

A.5 Conclusion

Ce travail de thèse s'est concentré sur une nouvelle méthode qui permet d'identifier, à partir des mesures de la plate-forme, les facteurs qui déterminent le *temps de réponse*. En particulier, les contributions de la thèse ont proposé et évalué les éléments suivants.

Contribution 1 (Section A.4.1). Le DFF HLS, créé à partir de la combinaison de la programmation fonctionnelle CAPH basée sur la génération DF et de l'outil de gestion du chemin de données matériel reconfigurable MDC, permet de concevoir un système avec une *fonctionnalité* paramétrique ou, plus précisément, avec une de ses tâches paramétriques mise en œuvre en tant que CGR IP. Par rapport aux approches à l'état de l'art, DFF HLS conduit à i) une prévisibilité précoce en termes de *temps de réponse* au niveau IP, et ii) une solution orientée vers les *fonctionnalités* synthétisable pour différentes plateformes (FPGAs et ASIC). En co-processing avec d'autres processeurs polyvalents, le IP généré peut être intégré dans un MPSoC en utilisant le cadre d'*adaptation* proposé pour gérer dynamiquement des *fonctionnalités* paramétriques mises en œuvre par le biais de spécifications matérielles et logicielles basées sur une représentation du flux de données de l'application.

Contribution 2 (Section A.4.2). L'outil PathFinder est proposé et vise à comprendre les facteurs qui déterminent le *temps de réponse* au niveau du système pour les *fonctionnalités* paramétriques exécutées sur des MPSoCs hétérogènes. Par rapport aux approches traditionnelles, PF propose un point de vue différent pour identifier l'activité de chronométrage par une approximation de la succession des tâches causant le *temps de réponse*, appelé Longest-Latency Path.

Contribution 3 (Section A.4.3). Une évaluation d'une approche basée sur le MoA pour estimer la *latence d'exécution du système* des *fonctionnalités* caractérisées par le LLPs comme activité d'application est proposée. Bien que la méthode soit loin de fournir une solution complète en termes de généralisation aux mappages, aux paramètres d'application, à l'application et aux architectures, elle représente la première tentative de construction d'un modèle élastique visant à soutenir la gestion du temps d'exécution dans le contexte complexe de l'analyse du temps MPSoC.

List of Figures

List of Tables

Acronyms

ALU Arithmetic Logic Unit. 30, 31

API Application Programming Interface. 26, 28

ASIC Application-Specific Integrated Circuit. 28–30, 33, 34, 54, 56, 129, 137, 147

AVC Advanced Video Coding. 74

AWS Amazon Web Services. 52

BCET Best-Case Execution Time. 37

BDF Boolean DF. 24, 26

BP Bin-Packing. 42

BPDF Boolean Parametric DF. 24, 25

CAL Cal Actor Language. 32, 46, 56

CAPH Caph just Ain't Plain HDL. 5, 33, 34, 44, 45, 56, 57, 71, 72, 74, 129, 130, 141, 142, 147, 149

CC clock cycle. 55, 74, 76, 77, 153

CERBERO Cross-layer modEl-based fRamework for multi-oBjective dEsign of Reconfigurable systems in unceRtain hybRid envirOnments. 14, 16, 44, 71, 80, 81, 88, 130, 142, 149, 150

CFDF Core Functional DF. 24, 25

CFG Control-Flow Graph. 35, 36, 149

CG Coarse-Grained. 17, 29, 30, 44, 138, 149, 155

CGR CG Reconfigurable. 5, 6, 17, 19, 20, 29–34, 46, 49, 51, 52, 54, 56, 57, 65, 66, 69–73, 75, 77, 79, 80, 83, 87, 88, 129, 130, 132, 135, 137, 138, 140–142, 147, 153

CGRA Coarse-Grained Reconfigurable Architecture. 5, 27, 29–33, 45, 46, 49, 81, 138, 149

CN Communication Node. 115–117, 145, 146

CP Critical Path. 6, 7, 60, 61, 63–65, 90–101, 104, 107, 110, 111, 132, 143–145, 150, 153

CPN C for Process Networks. 25, 26

CPS Cyber-Physical System. 5–7, 13–15, 37, 49, 51–64, 66, 68, 70–72, 74, 76, 78, 80, 82, 84, 86, 88, 133, 134, 139–141, 149

CPU Central Processing Unit. 27–29, 32, 33, 66, 68, 137, 141

CSDF Cyclo-Static DF. 24–26, 59

DAET Deterministic Actor Execution Time. 6, 58, 60, 61, 63, 69

DAG Directed Acyclic Graph. 6, 51, 58–63, 65, 69, 81, 88–90, 92, 96–98, 100–102, 105, 117, 119, 132, 140, 143, 150, 153, 154, 159

DF dataflow. 5, 17, 20–26, 33, 34, 43–46, 49, 53, 56, 57, 59, 70–75, 78–81, 83, 84, 86–90, 98, 113, 123, 129, 130, 136, 137, 144, 146, 147, 150, 153, 155–157, 159

DFF DataFlow-Functional. 5, 6, 34, 54–57, 71–79, 81, 87, 129, 130, 139, 141, 142, 147, 150, 151, 153

DFN dataflow network. 44, 45, 57, 72, 74, 80, 81, 141, 142

DLS Dumped-Least Squares. 83, 86, 87, 150

DPN Dataflow Process Network. 23, 24, 26, 32, 45, 48

DSA Domain-Specific Accelerator. 5, 6, 51, 52, 54, 56, 57, 71–73, 75, 77, 80, 83, 87, 141, 142

DSE Design Space Exploration. 20–22, 25, 32, 33, 49, 54, 55, 61, 71, 77, 120, 136, 139, 153

DSP Digital Signal Processor. 28, 31, 138

EIDF Enable-Invoke DF. 24, 25

FCFS First-Come First-Served. 38

FG Fine-Grained. 29, 30, 32, 138, 149, 156

FGR FG Reconfigurable. 30, 32, 156

FGRA FGR Architecture. 30

FI Full Integration. 39–41, 43, 44, 138

FIFO First In, First Out data queue. 21, 23, 48, 53, 57–59, 72, 75, 80, 137, 141

FitOptiVis From the cloud to the edge – smart IntegraTion and OPtimisation Technologies for highly efficient Image and VIdeo processing Systems. 14

FM Fidelity Metric. 67, 126, 154

FPGA Field-Programmable Gate Array. 29–34, 52, 54, 56, 57, 66, 76–80, 87, 129, 137, 138, 147, 150, 153

FSM Finite State Machine. 24

FU functional unit. 31, 138

GPU Graphics Processing Unit. 28, 33, 66, 68, 141

GRT Global Runtime. 48

GS Gantt Similarity. 7, 64, 108–110, 145

HCL Hardware Component Library. 72

HDF Heterochronous DF. 24, 26

HDL Hardware Description Language. 21, 32–34, 44, 54–56, 72, 87, 140, 155

HEVC High Efficiency Video Coding. 74, 142

HLS High-Level Synthesis. 5, 6, 32–34, 45, 51, 52, 54–57, 71–79, 81, 87, 129, 130, 139, 141, 142, 147, 150, 151, 153

HPC High Performance Computing. 33, 51, 52, 66

HPEC High Performance Embedded Computing. 66

HW hardware. 5, 6, 13–17, 19–22, 24, 26–30, 32–34, 36, 38–46, 48–58, 60, 62, 64–84, 86–88, 90, 91, 98, 113, 114, 116, 118, 123, 127, 129, 130, 132, 139, 149–151, 154

IDE Integrated Development Environment. 33

IK Inverse Kinematics. 80, 83

ILP Integer Linear Programming. 42

IoU Intersection over Union. 64, 145

IP Intellectual Property. 33, 46, 51, 54–56, 69, 74, 76, 83, 87, 88, 129–132, 139, 140, 147

IS Integration into Schedulability Analysis. 41–44, 138

ISA Instruction Set Architecture. 27, 28, 36, 96

ISS Instruction Set Simulator. 20, 22, 36, 136

KPI Key Performance Indicator. 15, 19–22, 29, 57, 58, 66–68, 113–115, 127, 134–136, 140

KPN Kahn Process Network. 23–26, 32

LLP Longest-Latency Path. 6–8, 43, 58, 62, 64, 69, 89–91, 94–98, 109, 111, 113–127, 130, 132, 140, 143–147, 150, 151, 153

LRT Local Runtime. 48

LSLA Linear System-Level Architecture Model. 7, 17, 68, 69, 115–117, 123, 124, 126, 132, 145, 146, 151

MAI Multiple Applications Information. 120, 121

MDC Multi-Dataflow Composer. 5, 6, 16, 33, 34, 45, 46, 56, 71–74, 78–80, 82, 83, 87, 88, 129, 130, 141, 142, 147, 149

MDG Multi-Dataflow Generator. 72

MILP Mixed-Integer Linear Programming. 42

MMI Multiple Mappings Information. 119–121, 125, 126, 151

MoA Model of Architecture. 7, 8, 15–17, 21, 43, 44, 49, 51, 58, 67–71, 88, 113–127, 130, 132, 135, 141, 145–147, 150, 151

MoC Model of Computation. 5, 17, 19–26, 32, 33, 44, 45, 48, 49, 53, 58, 65, 68, 78, 80, 89, 98, 123, 129, 135–137, 143, 144, 146, 153

MP Merging Process. 45

MPSoC Multiprocessor SoC. 5, 7, 13, 16, 17, 23, 26, 34, 35, 37–39, 41, 43, 44, 49, 58, 60, 62, 64, 66, 69, 89, 90, 95–97, 107, 111, 113, 129–133, 135, 138, 140, 143, 147, 150

MS Mapping and Scheduling. 42–44, 139

MSI Multiple Scenarios Information. 119–121

NDA non-disclosure agreement. 129

NP Nondeterministic Polynomial. 39, 42

ORCC Open RVC-CAL Compiler. 32, 33

PC Platform Composer. 46, 72

PE Processing Element. 13, 15–17, 19, 20, 25, 27, 29–35, 37–40, 42, 43, 46, 47, 49, 52, 57, 61–63, 65, 66, 69, 72–74, 78, 80, 81, 90, 91, 95, 96, 98–100, 104, 110, 113, 115–117, 119, 121–125, 129–131, 133, 135–138, 141, 142, 145, 146, 154

PF PathFinder. 6, 8, 23, 43, 44, 49, 58, 60, 62–64, 69–71, 89, 90, 92–94, 96, 98, 100, 102, 104, 106–111, 115, 121, 122, 124, 125, 127, 130, 132, 137, 140, 143–147, 150, 151

PiSDF Parameterized and Interfaced SDF. 24–26, 47–49, 59, 80, 81, 84, 85, 87, 89, 98–100, 102, 105, 131, 137, 143, 144, 150, 153, 154

PoC Proof of Concept. 6, 72, 80, 81, 83, 88, 132

PREESM Parallel and Real-time Embedded Executives Scheduling Method. 5, 25, 26, 47, 80, 84, 86, 91, 98, 110, 149, 150

PSDF Parameterized SDF. 24, 26

RAM Random Access Memory. 98

RISC Reduced Instruction Set Computer. 32

RR Round-Robin. 37, 38, 41, 149

RTL Register Transfer Level. 21, 33, 54, 55, 77, 136

S-LAM System-Level Architecture Model. 26, 47, 131

SADF Scenario-Aware DF. 24, 26

SB Switching Boxe. 73

SDF Synchronous DF. 21, 23–26, 56–59, 131, 159

SIFT Scale Invariant Feature Transform. 101, 102, 105, 107, 108, 110, 144, 151, 154

SMI Single Mapping Information. 118, 120, 121, 125, 126, 151

SoC System on Chip. 13, 80, 158

SPDF Schedulable Parametric DF. 24, 25

SPIDER Synchronous Parameterized and Interfaced Dataflow Embedded Runtime. 5, 16, 26, 48, 49, 71, 78–80, 83, 86–88, 142, 149, 150

sr single-rate. 47, 59, 81, 159

srDAG single-rate DAG. 59, 64, 68, 90–94, 98, 99, 101, 103, 110, 124, 145, 146

SW software. 5, 6, 13–17, 19–22, 24, 26, 28–30, 32, 34, 36, 38–40, 42–44, 46, 48–52, 54, 56–72, 74, 76–84, 86–88, 90, 91, 127, 129–131, 150

TDMA Time Division Multiple Access. 37, 38, 41, 149

TI Temporal Isolation. 40, 41, 43, 44, 138

TLM Transaction-Level Modeling. 21, 136

UML Unified Modeling Language. 21, 136

VPU Vision Processing Unit. 52

WCET Worst-Case Execution Time. 22, 35–37, 39–41, 138, 149

WCRT Worst-Case Response Time. 37, 41

XDF eXtensible Data Format. 72

GLOSSARY

acceleration Usage of custom hardware specially implemented to perform one or more functionalities more efficiently compared to the software running on a general-purpose unit. 17, 20, 29, 32, 34, 54, 66, 87, 153

adaptation Capability of reacting to different inputs coming from the environment, the user or the system state with a change of scheduling and mapping of the application. 6, 14–17, 27, 29, 65, 66, 69–71, 78, 80–83, 88, 111, 127, 129, 130, 132, 149, 150

elasticity Ability of an analysis to be changed in order to effectively achieve the same result by meeting the requirements with a certain margin. 16, 22, 120

flexibility Ability of a system to change or to be changed in order to effectively perform a computation according to the general-purpose input specifications. 13–16, 22, 27–30, 36, 49, 52, 73, 87, 91, 129, 149

functionality Set of operations or tasks satisfying the design needs that can be implemented in software and/or hardware. 13–17, 19–23, 25–35, 43, 45, 47, 49, 54, 55, 59, 65–67, 69, 80, 81, 84, 87, 88, 90, 113, 118, 127, 129–132, 150, 161

interference Set of processes/events that prevent the ideal execution of a functionality according to its representation. 7, 37, 39–41, 61, 64, 65, 90, 91, 94–97, 99, 101, 104, 107–111, 116, 132, 150

latency See response time. 5, 6, 20–23, 29, 33, 36, 38, 47, 51, 52, 54–58, 60–64, 66, 68–77, 87, 90–92, 94–96, 98, 100, 104, 108–110, 149, 153

operating mode Condition in which specific operations or tasks of a functionality are carried out depending on its configuration parameters. 13, 14, 19, 29, 41, 49, 88, 91, 113, 117, 129, 161, 162

reconfiguration Quality of a hardware and/or software system to perform more functionalities or different operating modes of one and/or more functionalities. 13–17, 21, 23–25, 29, 30, 32, 33, 39, 49, 56, 57, 65, 66, 69, 70, 72, 73, 79, 80, 87, 88, 91, 127, 129, 131, 149

response time Execution time required to complete an iteration of the functionality considered in a specific operating mode. 7, 13, 15–17, 19, 20, 22, 35, 36, 41, 43, 49, 51, 54–58, 60, 64–66, 68–71, 76, 79, 88–90, 94, 95, 98–101, 104, 107–111, 113–120, 122, 124–127, 129–132, 151, 161, 162

scenario See operating mode. 6, 24, 29, 36, 38, 47, 90, 91, 98, 100–105, 107–110, 113, 114, 117–121, 123–125, 127, 151, 153, 154

system execution latency See response time. 17, 23, 55, 57, 58, 60, 62, 65, 68, 69, 71, 89, 90, 110, 113, 119, 127, 130

Personal Publications

[RPP17] C. Rubattu, F. Palumbo, and M. Pelcat, « Adaptive software-augmented hardware reconfiguration with dataflow design automation », *in*: *2017 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, 2017, DOI: 10.1109/RECONFIG.2017.8279772.

[Rub+19] C. Rubattu et al., « Dataflow-Functional High-Level Synthesis for Coarse-Grained Reconfigurable Accelerators », *in*: *IEEE Embedded Systems Letters* 11.*3* (2019), pp. 69–72, DOI: 10.1109/LES.2018.2882989 (cit. on pp. 34, 72, 141).

[Rub18] C. Rubattu, « Dataflow-based Adaptation Framework with Coarse-Grained Reconfigurable Accelerators », *in*: *Cyber-Physical Systems Workshop (CPSWS)*, 2018 (cit. on p. 77).

[Fan+19a] L. Fanni et al., « A Dataflow Implementation of Inverse Kinematics on Reconfigurable Heterogeneous MPSoC », *in*: *Cyber-Physical Systems Workshop (CPSWS)*, 2019 (cit. on p. 83).

[Fan+19b] T. Fanni et al., « Run-time Performance Monitoring of Heterogenous Hw/Sw Platforms Using PAPI », *in*: *FSP Workshop 2019; Sixth International Workshop on FPGAs for Software Programmers*, 2019 (cit. on pp. 80, 82).

[Pal+19] F. Palumbo et al., « Hardware/Software Self-adaptation in CPS: The CERBERO Project Approach », *in*: *Embedded Computer Systems: Architectures, Modeling, and Simulation*, ed. by D. N. Pnevmatikatos, M. Pelcat, and M. Jung, Cham: Springer International Publishing, 2019, pp. 416–428, ISBN: 978-3-030-27562-4, DOI: 10.1007/978-3-030-27562-4_30.

[Pay+19] S. Payvar et al., « Extending Architecture Modeling for Signal Processing towards GPUs », *in*: *2019 27th European Signal Processing Conference (EUSIPCO)*, 2019, pp. 1–5, DOI: 10.23919/EUSIPCO.2019.8903094 (cit. on pp. 68, 69, 141).

[Sau+20a] C. Sau et al., « Feasibility Study and Porting of the Damped Least Square Algorithm on FPGA », *in*: *IEEE Access* (2020), DOI: 10.1109/ACCESS.2020.3025367.

[Sau+20b] C. Sau et al., « The Multi-Dataflow Composer Tool: an open-source tool suite for Optimized Coarse-Grain Reconfigurable Hardware Accelerators and Platform Design », *in*: *Microprocessors and Microsystems* (2020), DOI: 10.1016/j.micpro.2020.103326 (cit. on p. 83).

Bibliography

[AA18] M. Ammar and M. Abid, « Heterogeneity of abstractions in EDA tools: Reviewing models of computation for many-core systems targeting intensive signal processing applications », *in*: *Microprocessors and Microsystems* 59 (2018), pp. 1–14, DOI: 10.1016/j.micpro.2018.03.001 (cit. on pp. 20, 22, 136).

[ABA16] M. Ammar, M. Baklouti, and M. Abid, « The Performance-Energy Tradeoff in Embedded Systems Design: A Survey of Existing Design Space Exploration Tools and Trends », *in*: *International Journal of Computer Science and Information Security (IJCSIS)* 14.5 (2016), pp. 381–391 (cit. on pp. 20, 22, 136).

[Abe+13] A. Abel et al., « Impact of Resource Sharing on Performance and Performance Prediction: A Survey », *in*: *CONCUR 2013 – Concurrency Theory*, ed. by P. R. D'Argenio and H. Melgratti, 2013, pp. 25–43, DOI: 10.1007/978-3-642-40184-8_3 (cit. on pp. 38, 39).

[Acc] Accellera Systems Initiative, *SystemC*, URL: https://www.accellera.org/downloads/standards/systemc (cit. on pp. 21, 136).

[AG17] I. Assayad and A. Girault, « Adaptive Mapping for Multiple Applications on Parallel Architectures », *in*: *Ubiquitous Networking*, ed. by E. Sabir et al., 2017, pp. 584–595, DOI: 10.1007/978-3-319-68179-5_51 (cit. on p. 66).

[And+18] B. Andersson et al., « Schedulability Analysis of Tasks with Corunner-Dependent Execution Times », *in*: *ACM Transactions on Embedded Computing Systems* 17.3 (2018), DOI: 10.1145/3203407 (cit. on p. 41).

[AP14] A. Alhammad and R. Pellizzoni, « Schedulability Analysis of Global Memory-Predictable Scheduling », *in*: *Proceedings of the 14th International Conference on Embedded Software*, 2014, DOI: 10.1145/2656045.2656070 (cit. on p. 37).

[AS19] A. Akram and L. Sawalha, « A Survey of Computer Architecture Simulation Techniques and Tools », *in*: *IEEE Access* 7 (2019), DOI: 10.1109/ACCESS.2019.2917698 (cit. on pp. 20, 136).

[Axe+14] P. Axer et al., « Building Timing Predictable Embedded Systems », *in*: *ACM Transactions on Embedded Computing Systems* 13.*4* (2014), DOI: 10.1145/2560033 (cit. on p. 37).

[Bac07] J. Backus, *Can programming be liberated from the von Neumann style?: a functional style and its algebra of programs*, ACM, 2007, DOI: 10.1145/359576.359579 (cit. on p. 52).

[Bak+12] S. Bak et al., « Memory-Aware Scheduling of Multicore Task Sets for Real-Time Systems », *in*: *2012 IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, 2012, DOI: 10.1109/RTCSA.2012.48 (cit. on p. 40).

[Bar17] J. Barr, *EC2 F1 Instances with FPGAs - Now Generally Available*, 2017, URL: https://aws.amazon.com/blogs/aws/ec2-f1-instances-with-fpgas-now-generally-available/ (cit. on p. 52).

[BB00] N. Bambha and S. S. Bhattacharyya, « A joint power/performance optimization algorithm for multiprocessor systems using a period graph construct », *in*: *Proceedings 13th International Symposium on System Synthesis*, 2000, DOI: 10.1109/ISSS.2000.874034 (cit. on pp. 67, 141, 146).

[BB01] B. Bhattacharya and S. S. Bhattacharyya, « Parameterized dataflow modeling for DSP systems », *in*: *IEEE Transactions on Signal Processing* 49.*10* (2001), pp. 2408–2421, DOI: 10.1109/78.950795 (cit. on p. 24).

[Bea+10] C. Beaumin et al., « A coarse-grain reconfigurable hardware architecture for RVC-CAL-based design », *in*: *2010 Conference on Design and Architectures for Signal and Image Processing (DASIP)*, 2010, DOI: 10.1109/DASIP.2010.5706259 (cit. on pp. 32, 34).

[Beb+13] V. Bebelis et al., « BPDF: A statically analyzable dataflow model with integer and boolean parameters », *in*: *2013 Proceedings of the International Conference on Embedded Software (EMSOFT)*, 2013, DOI: 10.1109/EMSOFT.2013.6658581 (cit. on p. 25).

[Bec+16] M. Becker et al., « Contention-Free Execution of Automotive Applications on a Clustered Many-Core Platform », *in*: *2016 28th Euromicro Conference on Real-Time Systems (ECRTS)*, 2016, DOI: 10.1109/ECRTS.2016.14 (cit. on p. 42).

[Bil+96] G. Bilsen et al., « Cycle-static dataflow », *in*: *IEEE Transactions on Signal Processing* 44.*2* (Feb. 1996), pp. 397–408, DOI: 10.1109/78.485935 (cit. on pp. 24, 59).

[BMd12] B. Bodin, A. Munier-Kordon, and B. D. de Dinechin, « K-Periodic schedules for evaluating the maximum throughput of a Synchronous Dataflow graph », *in*: *2012 International Conference on Embedded Computer Systems (SAMOS)*, 2012, DOI: 10.1109/SAMOS.2012.6404169 (cit. on p. 25).

[BMJ13] E. Bezati, M. Mattavelli, and J. Janneck, « High-level synthesis of dataflow programs for signal processing systems », *in*: *Image and Signal Processing and Analysis (ISPA), 2013 8th International Symposium on*, 2013, DOI: 10.1109/ISPA.2013.6703837 (cit. on p. 32).

[Bon+12] F. Boniol et al., « Deterministic Execution Model on COTS Hardware », *in*: *Architecture of Computing Systems – ARCS 2012*, 2012, DOI: 10.1007/978-3-642-28293-5_9 (cit. on p. 40).

[Bon+19] V. Bonifaci et al., « A Generalized Parallel Task Model for Recurrent Real-Time Processes », *in*: *ACM Transactions on Parallel Computing* 6.*1* (2019), DOI: 10.1145/3322809 (cit. on p. 59).

[Bra+18] G. M. Bragg et al., « An application- and platform-agnostic control and monitoring framework for multicore systems », *in*: *3rd International Conference on Pervasive and Embedded Computing*, 2018 (cit. on p. 66).

[Buc93] J. Buck, « Scheduling dynamic dataflow graphs with bounded memory using the token flow model », PhD thesis, Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, 1993 (cit. on p. 24).

[CC18] T. Carle and H. Cassé, « Reducing Timing Interferences in Real-Time Applications Running on Multicore Architectures », *in*: *18th International Workshop on Worst-Case Execution Time Analysis (WCET 2018)*, 2018, DOI: 10.4230/OASIcs.WCET.2018.3 (cit. on p. 41).

[CF16] A. Cilardo and E. Fusella, « Design automation for application-specific on-chip interconnects: A survey », *in*: *Integration* 52 (2016), pp. 102–121, DOI: 10.1016/j.vlsi.2015.07.017 (cit. on pp. 13, 133).

[CH02] K. Compton and S. Hauck, « Reconfigurable computing: a survey of systems and software », *in*: *ACM Computing Surveys* 34.*2* (2002), DOI: 10.1145/508352.508353 (cit. on pp. 29, 32, 137).

[Che+16] S. Cheng et al., « Many-Core Real-Time Task Scheduling with Scratchpad Memory », *in*: *IEEE Transactions on Parallel and Distributed Systems* (2016), DOI: 10.1109/TPDS.2016.2516519 (cit. on p. 42).

[Che05] P. Cheung, « Reconfigurable computing: architectures and design methods », *in*: *IEE Proceedings - Computers and Digital Techniques* 152 (2 2005), 193–207(14), DOI: 10.1049/ip-cdt:20045086 (cit. on pp. 29, 138).

[Chi+16] M. Chisholm et al., « Reconciling the Tension Between Hardware Isolation and Data Sharing in Mixed-Criticality, Multicore Systems », *in*: *2016 IEEE Real-Time Systems Symposium (RTSS)*, 2016, DOI: 10.1109/RTSS.2016.015 (cit. on p. 42).

[Cio+18] C. B. Ciobanu et al., « EXTRA: An Open Platform for Reconfigurable Architectures », *in*: *Proceedings of the 18th International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation*, 2018, DOI: 10.1145/3229631.3236092 (cit. on pp. 33, 34).

[CMJ13] S. Casale-Brunet, M. Mattavelli, and J. Janneck, « TURNUS: A design exploration framework for dataflow system design », *in*: *Circuits and Systems (ISCAS), 2013 IEEE International Symposium on*, 2013, DOI: 10.1109/ISCAS.2013.6571927 (cit. on p. 32).

[Dav+17] R. Davis et al., « An extensible framework for multicore response time analysis », *in*: *Real-Time Systems* 54 (2017), pp. 607–661, DOI: 10.1007/s11241-017-9285-4 (cit. on p. 41).

[Des+13] K. Desnos et al., « Pimm: Parameterized and interfaced dataflow metamodel for mpsocs runtime reconfiguration », *in*: *International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS)*, 2013, DOI: 10.1109/SAMOS.2013.6621104 (cit. on pp. 24, 59, 98, 143, 144).

[DLS12] P. Derler, E. A. Lee, and A. Sangiovanni Vincentelli, « Modeling Cyber–Physical Systems », *in*: *Proceedings of the IEEE* 100.*1* (2012), pp. 13–28, DOI: 10.1109/JPROC.2011.2160929 (cit. on pp. 13, 133).

[DN12] D. Dasari and V. Nelis, « An Analysis of the Impact of Bus Contention on the WCET in Multicores », *in*: *2012 IEEE 14th International Conference on High Performance Computing and Communication 2012 IEEE 9th International Conference on Embedded Software and Systems*, 2012, DOI: 10.1109/HPCC.2012.212 (cit. on p. 40).

[DS07] H. Dybdahl and P. Stenstrom, « An Adaptive Shared/Private NUCA Cache Partitioning Scheme for Chip Multiprocessors », *in*: *2007 IEEE 13th International Symposium on High Performance Computer Architecture*, 2007, DOI: 10.1109/HPCA.2007.346180 (cit. on p. 38).

[Fan+15] T. Fanni et al., « Automated Power Gating Methodology for Dataflow-based Reconfigurable Systems », *in*: *Proceedings of the 12th ACM International Conference on Computing Frontiers, CF'15*, 2015, DOI: 10.1145/2742854.2747285 (cit. on p. 46).

[Fan+16] T. Fanni et al., « Power and clock gating modelling in coarse grained reconfigurable systems », *in*: *Proceedings of the ACM International Conference on Computing Frontiers, CF'16*, 2016, DOI: 10.1145/2903150.2911713 (cit. on p. 46).

[FGP12] P. Fradet, A. Girault, and P. Poplavko, « SPDF: A schedulable parametric data-flow MoC », *in*: *2012 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2012, DOI: 10.1109/DATE.2012.6176572 (cit. on p. 25).

[FUU18] J. Freitag, S. Uhrig, and T. Ungerer, « Virtual Timing Isolation for Mixed-Criticality Systems », *in*: *30th Euromicro Conference on Real-Time Systems (ECRTS 2018)*, 2018, DOI: 10.4230/LIPIcs.ECRTS.2018.13 (cit. on p. 40).

[FW19] A. Fuchs and D. Wentzlaff, « The accelerator wall: Limits of chip specialization », *in*: *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2019, DOI: 10.1109/HPCA.2019.00023 (cit. on pp. 23, 29, 58, 59, 137, 140).

[Gam+08] A. Gamatié et al., « Synchronous Modeling and Analysis of Data Intensive Applications », *in*: *EURASIP Journal on Embedded Systems* 2008.*1* (2008), DOI: 10.1155/2008/561863 (cit. on p. 23).

[Gam+11] A. Gamatié et al., « A Model-Driven Design Framework for Massively Parallel Embedded Systems », *in*: *ACM Transactions on Embedded Computing Systems* 10.*4* (2011), DOI: 10.1145/2043662.2043663 (cit. on p. 21).

[Gau+13] T. Gautier et al., « XKaapi: A Runtime System for Data-Flow Task Programming on Heterogeneous Architectures », *in*: *2013 IEEE 27th International Symposium on Parallel and Distributed Processing*, 2013, DOI: 10.1109/IPDPS.2013.66 (cit. on p. 66).

[GBL99] A. Girault, Bilung Lee, and E. A. Lee, « Hierarchical finite state machines with multiple concurrency models », *in*: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 18.*6* (1999), pp. 742–760, DOI: 10.1109/43.766725 (cit. on p. 24).

[Ger+16] I. Gerostathopoulos et al., « Self-adaptation in software-intensive cyber–physical systems: From system goals to architecture configurations », *in*: *Journal of Systems and Software* 122 (2016), pp. 378–397, DOI: 10.1016/j.jss.2016.02.028 (cit. on p. 66).

[Gia+15] G. Giannopoulou et al., « Mixed-criticality scheduling on cluster-based many-cores with shared communication and storage resources », *in*: *Real-Time Systems* 52.*4* (2015), pp. 399–449, DOI: 10.1007/s11241-015-9227-y (cit. on p. 41).

[Gli+15] C. Glitia et al., « Progressive and Explicit Refinement of Scheduling for Multidimensional Data-Flow Applications Using UML MARTE », *in*: *Design Automation for Embedded Systems* 19.*1–2* (2015), pp. 1–33, DOI: 10.1007/s10617-014-9140-y (cit. on p. 21).

[GPK19] R. Giorgi, M. Procaccini, and F. Khalili, « AXIOM: A Scalable, Efficient and Reconfigurable Embedded Platform », *in*: *2019 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2019, pp. 480–485, DOI: 10.23919/DATE.2019.8715168 (cit. on pp. 13, 134).

[GPM14] M. García-Valls, D. Perez-Palacin, and R. Mirandola, « Time-Sensitive Adaptation in CPS through Run-Time Configuration Generation and Verification », *in*: *2014 IEEE 38th Annual Computer Software and Applications Conference*, 2014, DOI: 10.1109/COMPSAC.2014.55 (cit. on pp. 15, 134).

[Gra] M. Graphics, *Catapult High-Level Synthesis*, URL: https://www.mentor.com/hls-lp/catapult-high-level-synthesis/ (cit. on pp. 33, 34).

[Gra+15] G. Gracioli et al., « A Survey on Cache Management Mechanisms for Real-Time Embedded Systems », *in*: *ACM Computing Surveys* 48.*2* (2015), DOI: 10.1145/2830555 (cit. on p. 39).

[Gro] K. Group, *OpenCL*, URL: https://www.khronos.org/opencl/ (cit. on p. 28).

[Gua+11] N. Guan et al., « Schedulability analysis for non-preemptive fixed-priority multiprocessor scheduling », *in*: *Journal of Systems Architecture* 57.*5* (2011), pp. 536–546, DOI: 10.1016/j.sysarc.2010.08.003 (cit. on p. 37).

[Gus+10] A. Gustavsson et al., « Towards WCET Analysis of Multicore Architectures Using UPPAAL », *in*: *10th International Workshop on Worst-Case Execution Time Analysis (WCET 2010)*, 2010, DOI: 10.4230/OASIcs.WCET.2010.101 (cit. on p. 40).

[Har] Hardkernel co. Ltd, *ODROID-XU3*, URL: https://www.hardkernel.com/shop/odroid-xu3/ (cit. on pp. 98, 125, 144, 146).

[Har01] R. Hartenstein, « Coarse grain reconfigurable architecture (embedded tutorial) », *in*: *Proceedings of ASP-DAC 2001, Asia and South Pacific Design Automation Conference*, 2001, DOI: 10.1145/370155.370535 (cit. on pp. 30, 31, 138).

[HAR14] S. A. Haque, S. M. Aziz, and M. Rahman, « Review of Cyber-Physical System in Healthcare », *in*: *International Journal of Distributed Sensor Networks* 10.*4* (2014), p. 217415, DOI: 10.1155/2014/217415 (cit. on pp. 15, 134).

[Hen+11] J. Henkel et al., « Design and Architectures for Dependable Embedded Systems », *in*: *Proceedings of the Seventh IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, 2011, DOI: 10.1145/2039370.2039384 (cit. on p. 16).

[Heu+14] J. Heulot et al., « SPIDER: A Synchronous Parameterized and Interfaced Dataflow-based RTOS for multicore DSPS », *in*: *2014 6th European Embedded Design in Education and Research Conference (EDERC)*, 2014, DOI: 10.1109/EDERC.2014.6924381 (cit. on pp. 26, 48, 142).

[HKP17] M. Hassan, A. M. Kaushik, and H. Patel, « Predictable Cache Coherence for Multi-core Real-Time Systems », *in*: *2017 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2017, DOI: 10.1109/RTAS.2017.13 (cit. on p. 37).

[Hor+09] A. H. Hormati et al., « Flextream: Adaptive Compilation of Streaming Applications for Heterogeneous Architectures », *in*: *2009 18th International Conference on Parallel Architectures and Compilation Techniques*, 2009, DOI: 10.1109/PACT.2009.39 (cit. on p. 66).

[HP17] J. L. Hennessy and D. A. Patterson, *Computer Architecture, Sixth Edition: A Quantitative Approach*, 6th, San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2017, ISBN: 0128119055 (cit. on pp. 27, 28).

[HP18] M. Hassan and R. Pellizzoni, « Bounding DRAM Interference in COTS Heterogeneous MPSoCs for Mixed Criticality Systems », *in*: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 37.*11* (2018), pp. 2323–2336, DOI: 10.1109/TCAD.2018.2857379 (cit. on p. 65).

[HPB17] M. Han, J. Park, and W. Baek, « CHRT: A criticality- and heterogeneity-aware runtime system for task-parallel applications », *in*: *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*, 2017, DOI: 10.23919/DATE.2017.7927126 (cit. on p. 66).

[HSM03] P. Heysters, G. Smit, and E. Molenkamp, « A Flexible and Energy-Efficient Coarse-Grained Reconfigurable Architecture for Mobile Systems », *in*: *The Journal of Supercomputing* 26.*3* (2003), pp. 283–308, DOI: 10.1023/A:1025699015398 (cit. on p. 30).

[Hsu+07] C.-J. Hsu et al., « Efficient Simulation of Critical Synchronous Dataflow Graphs », *in*: *ACM Transactions on Design Automation of Electronic Systems* 12.*3* (2007), DOI: 10.1145/1255456.1255458 (cit. on p. 59).

[Ino+10] Q. Inoue K.and Zhao et al., « A Variable-Grain Logic Cell and Routing Architecture for a Reconfigurable IP Core », *in*: *ACM Transactions on Reconfigurable Technology and Systems* 4.*1* (2010), 5:1–5:24, DOI: 10.1145/1857927.1857932 (cit. on p. 31).

[Int] Intel, *Intel FPGA SDK for OpenCL*, URL: https://www.altera.com/products/design-software/embedded-software-developers/opencl/ (cit. on pp. 33, 34, 52, 56).

[JHH15] M. Jacobs, S. Hahn, and S. Hack, « WCET Analysis for Multi-Core Processors with Shared Buses and Event-Driven Bus Arbitration », *in*: *Proceedings of the 23rd International Conference on Real Time and Networks Systems*, 2015, DOI: 10.1145/2834848.2834872 (cit. on p. 40).

[JHH16] M. Jacobs, S. Hahn, and S. Hack, « A Framework for the Derivation of WCET Analyses for Multi-core Processors », *in*: *2016 28th Euromicro Conference on Real-Time Systems (ECRTS)*, 2016, DOI: 10.1109/ECRTS.2016.19 (cit. on p. 40).

[JIP10] H. Javaid, A. Ignjatovic, and S. Parameswaran, « Fidelity metrics for estimation models », *in*: *2010 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2010, DOI: 10.1109/ICCAD.2010.5653959 (cit. on pp. 67, 141, 146).

[Jon+18] R. Jongerius et al., « Analytic Multi-Core Processor Model for Fast Design-Space Exploration », *in*: *IEEE Transactions on Computers* 67.*6* (June 2018), pp. 755–770, DOI: 10.1109/TC.2017.2780239 (cit. on p. 23).

[KA99] Y.-K. Kwok and I. Ahmad, « Static Scheduling Algorithms for Allocating Directed Task Graphs to Multiprocessors », *in*: *ACM Computing Surveys* 31.*4* (1999), pp. 406–471, DOI: 10.1145/344588.344618 (cit. on pp. 47, 98).

[Kah74] G. Kahn, « The semantics of a simple language for parallel programming », *in*: *In Information Processing* 74 (1974), pp. 471–475 (cit. on p. 23).

[Kal] Kalray, *MPPA AccessCore*, URL: https://www.kalrayinc.com/IMG/pdf/FLYER_MPPA_ACCESSCORE-2.pdf (cit. on p. 25).

[KAL11] T. Kempf, G. Ascheid, and R. Leupers, « Multiprocessor Systems on Chip: Design Space Exploration », *in*: 2011, DOI: 10.1007/978-1-4419-8153-0 (cit. on p. 60).

[Kel+14] T. Kelter et al., « Static analysis of multi-core TDMA resource arbitration delays », *in*: *Real-Time Systems* 50 (2014), pp. 185–229, DOI: 10.1007/s11241-013-9189-x (cit. on p. 37).

[Kie+97] B. Kienhuis et al., « An approach for quantitative analysis of application-specific dataflow architectures », *in*: *Proceedings IEEE International Conference on Application-Specific Systems, Architectures and Processors*, 1997, DOI: 10.1109/ASAP.1997.606839 (cit. on p. 68).

[Kim+16] H. Kim et al., « Bounding and reducing memory interference in COTS-based multi-core systems », *in*: *Real-Time Systems* 52.*3* (2016), pp. 356–395, DOI: 10.1007/s11241-016-9248-1 (cit. on p. 41).

[KM15] T. Kelter and P. Marwedel, « Parallelism Analysis: Precise WCET Values for Complex Multi-Core Systems », *in*: *Formal Techniques for Safety-Critical Systems*, ed. by C. Artho and P. C. Ölveczky, 2015, DOI: 10.1007/978-3-319-17581-2_10 (cit. on p. 40).

[LCK16] P. Leitão, A. W. Colombo, and S. Karnouskos, « Industrial automation based on cyber-physical systems technologies: Prototype implementations and challenges », *in*: *Computers in Industry* 81 (2016), Emerging ICT concepts for smart, safe and sustainable industrial systems, pp. 11–25, DOI: 10.1016/j.compind.2015.08.004 (cit. on p. 14).

[Leu+17] R. Leupers et al., « MAPS: A Software Development Environment for Embedded Multicore Applications », *in*: *Handbook of Hardware/Software Codesign*, ed. by S. Ha and J. Teich, Springer Netherlands, 2017, pp. 917–949, DOI: 10.1007/978-94-017-7267-9_2 (cit. on p. 25).

[Li+17] Y. Li et al., « Energy Optimization With Dynamic Task Scheduling Mobile Cloud Computing », *in*: *IEEE Systems Journal* 11.*1* (2017), pp. 96–105, DOI: 10.1109/JSYST.2015.2442994 (cit. on p. 59).

[LIP] LIP6, *SoCLib*, URL: http://www.soclib.fr/trac/dev (cit. on p. 21).

[Lit61] J. D. C. Little, « A Proof for the Queuing Formula: $L = \Lambda W$ », *in*: *Operations Research* 9.*3* (1961), pp. 383–387, DOI: 10.1287/opre.9.3.383 (cit. on pp. 89, 143).

[LM87] E. A. Lee and D. G. Messerschmitt, « Synchronous data flow », *in*: *Proceedings of the IEEE* 75.*9* (1987), pp. 1235–1245, DOI: 10.1109/PROC.1987.13876 (cit. on pp. 21, 23, 58).

[LP95] E. Lee and T. Parks, « Dataflow process networks », *in*: *Proceedings of the IEEE* 83.*5* (1995), pp. 773–801, DOI: 10.1109/5.381846 (cit. on p. 23).

[Lv+10] M. Lv et al., « Combining Abstract Interpretation with Model Checking for Timing Analysis of Multicore Software », *in*: *2010 31st IEEE Real-Time Systems Symposium*, 2010, DOI: 10.1109/RTSS.2010.30 (cit. on p. 40).

[Lv+16] M. Lv et al., « A Survey on Static Cache Analysis for Real-Time Systems », *in*: *Leibniz Transactions on Embedded Systems* 3.*1* (2016), 05–1-05:48, DOI: 10.4230/LITES-v003-i001-a005 (cit. on p. 36).

[LX04] E. Lee and Y. Xiong, « A behavioral type system and its application in Ptolemy II », *in*: *Formal Aspects of Computing* 16 (2004), pp. 210–237, DOI: 10.1007/s00165-004-0043-8 (cit. on pp. 19, 135).

[Mad+18] D. Madroñal et al., « Automatic Instrumentation of Dataflow Applications Using PAPI », *in*: *Proceedings of the 15th ACM International Conference on Computing Frontiers, CF'18*, 2018, DOI: 10.1145/3203217.3209886 (cit. on p. 80).

[Mai+19] C. Maiza et al., « A Survey of Timing Verification Techniques for Multi-Core Real-Time Systems », *in*: *ACM Computing Surveys* 52.*3* (2019), DOI: 10.1145/3323212 (cit. on pp. 37, 39, 41, 138).

[Man+17] R. Mancuso et al., « WCET Derivation under Single Core Equivalence with Explicit Memory Budget Assignment », *in*: *29th Euromicro Conference on Real-Time Systems (ECRTS 2017)*, 2017, DOI: 10.4230/LIPIcs.ECRTS.2017.3 (cit. on p. 40).

[Mat+19] J. Matejka et al., « Combining PREM compilation and static scheduling for high-performance and predictable MPSoC execution », *in*: *Parallel Computing* 85 (2019), pp. 27–44, DOI: 10.1016/j.parco.2018.11.002 (cit. on pp. 22, 137).

[Mel+15] A. Melani et al., « Response-Time Analysis of Conditional DAG Tasks in Multiprocessor Systems », *in*: *2015 27th Euromicro Conference on Real-Time Systems*, 2015, DOI: 10.1109/ECRTS.2015.26 (cit. on p. 59).

[Men+17] C. Menard et al., « System simulation with gem5 and SystemC: The keystone for full interoperability », *in*: *International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS)*, 2017, DOI: 10.1109/SAMOS.2017.8344612 (cit. on pp. 20, 21, 136).

[Mit17] S. Mittal, « A Survey of Techniques for Cache Partitioning in Multicore Processors », *in*: *ACM Computing Surveys* 50.*2* (2017), DOI: 10.1145/3062394 (cit. on p. 39).

[ML02] P. K. Murthy and E. A. Lee, « Multidimensional synchronous dataflow », *in*: *IEEE Transactions on Signal Processing* 50.*8* (2002), pp. 2064–2079, DOI: 10.1109/TSP.2002.800830 (cit. on p. 23).

[MTT18] T. Mitra, J. Teich, and L. Thiele, « Time-Critical Systems Design: A Survey », *in*: *IEEE Design Test* 35.*2* (2018), pp. 8–26, DOI: 10.1109/MDAT.2018.2794204 (cit. on p. 35).

[Nan+16] R. Nane et al., « A Survey and Evaluation of FPGA High-Level Synthesis Tools », *in*: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 35.*10* (2016), pp. 1591–1604, DOI: 10.1109/TCAD.2015.2513673 (cit. on p. 56).

[NKS12] A. Niedermeier, J. Kuper, and G. Smit, « Dataflow-based reconfigurable architecture for streaming applications », *in*: *System on Chip (SoC), 2012 International Symposium on*, 2012, DOI: 10.1109/ISSoC.2012.6376365 (cit. on p. 31).

[Nog16] E. Nogues, « Energy optimization of Signal Processing on MPSoCs and its Application to Video Decoding », PhD thesis, INSA de Rennes, 2016 (cit. on pp. 74, 75, 142).

[Now+13] T. Nowatzki et al., « A General Constraint-Centric Scheduling Framework for Spatial Architectures », *in*: *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2013, DOI: 10.1145/2491956.2462163 (cit. on p. 59).

[Now+14] J. Nowotsch et al., « Multi-core Interference-Sensitive WCET Analysis Leveraging Runtime Resource Capacity Enforcement », *in*: *2014 26th Euromicro Conference on Real-Time Systems*, 2014, DOI: 10.1109/ECRTS.2014.20 (cit. on p. 40).

[NP15] B. Nikolic and S. Petters, « Real-time application mapping for many-cores using a limited migrative model », *in*: *Real-Time Systems* 51 (2015), pp. 314–357, DOI: 10.1007/s11241-014-9217-5 (cit. on p. 42).

[NS16] K. Nagar and Y. N. Srikant, « Fast and Precise Worst-Case Interference Placement for Shared Cache Analysis », *in*: *ACM Transactions on Embedded Computing Systems* 15.*3* (2016), DOI: 10.1145/2854151 (cit. on p. 40).

[Nur+16] E. Nurvitadhi et al., « Accelerating Binarized Neural Networks: Comparison of FPGA, CPU, GPU, and ASIC », *in*: *2016 International Conference on Field-Programmable Technology (FPT)*, 2016, DOI: 10.1109/FPT.2016.7929192 (cit. on p. 29).

[OB18] K. O'Neal and P. Brisk, « Predictive Modeling for CPU, GPU, and FPGA Performance and Power Consumption: A Survey », *in*: *2018 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2018, DOI: 10.1109/ISVLSI.2018.00143 (cit. on p. 22).

[Obja] Object Management Group (OMG), *OMG System Modeling Language*, URL: https://www.omg.org/spec/SysML/1.4/ (cit. on pp. 21, 136).

[Objb] Object Management Group (OMG), *UML Profile for MARTE*, URL: https://www.omg.org/spec/MARTE/1.1/ (cit. on pp. 20, 21, 136).

[OLF17] D. Oehlert, A. Luppold, and H. Falk, « Bus-Aware Static Instruction SPM Allocation for Multicore Hard Real-Time Systems », *in*: *29th Euromicro Conference on Real-Time Systems (ECRTS 2017)*, 2017, DOI: 10.4230/LIPIcs.ECRTS.2017.1 (cit. on p. 41).

[Owe+08] J. D. Owens et al., « GPU Computing », *in*: *Proceedings of the IEEE* 96.*5* (2008), pp. 879–899, DOI: 10.1109/JPROC.2008.917757 (cit. on p. 28).

[Pal+12] F. Palumbo et al., « The multi-dataflow composer tool: Generation of on-the-fly reconfigurable platforms », *in*: *Journal of Real-Time Image Processing* 9 (Mar. 2012), pp. 1–17, DOI: 10.1007/s11554-012-0284-3 (cit. on p. 74).

[Pal+17] F. Palumbo et al., « Power-Awareness in Coarse-Grained Reconfigurable Multi-Functional Architectures: a Dataflow Based Strategy », *in*: *Journal of Signal Processing Systems* 87.*1* (2017), pp. 81–106, DOI: 10.1007/s11265-016-1106-9 (cit. on pp. 45, 141).

[PBR09] J. Piat, S. S. Bhattacharyya, and M. Raulet, « Interface-based hierarchy for synchronous data-flow graphs », *in*: *2009 IEEE Workshop on Signal Processing Systems*, 2009, DOI: 10.1109/SIPS.2009.5336240 (cit. on p. 24).

[PDM12] K. Paul, C. Dash, and M. Moghaddam, « reMORPH: A Runtime Reconfigurable Architecture », *in*: *Digital System Design (DSD), 2012 15th Euromicro Conference on*, 2012, DOI: 10.1109/DSD.2012.111 (cit. on p. 31).

[Pel+10] R. Pellizzoni et al., « Worst case delay analysis for memory interference in multicore systems », *in*: *2010 Design, Automation Test in Europe Conference Exhibition (DATE 2010)*, 2010, DOI: 10.1109/DATE.2010.5456952 (cit. on p. 41).

[Pel+13] M. Pelcat et al., « Dataflow Model of Computation », *in*: *Physical Layer Multi-Core Prototyping: A Dataflow-Based Approach for LTE eNodeB*, 2013, pp. 53–75, DOI: 10.1007/978-1-4471-4210-2_3 (cit. on pp. 23, 60).

[Pel+14] M. Pelcat et al., « Preesm: A dataflow-based rapid prototyping framework for simplifying multicore DSP programming », *in*: *2014 6th European Embedded Design in Education and Research Conference (EDERC)*, 2014, DOI: 10.13140/2.1.1819.0729 (cit. on pp. 25, 47, 98).

[Pel+16] M. Pelcat et al., « Design productivity of a high level synthesis compiler versus HDL », *in*: *2016 International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation (SAMOS)*, 2016, DOI: 10.1109/SAMOS.2016.7818341 (cit. on p. 52).

[Pel+18] M. Pelcat et al., « Reproducible Evaluation of System Efficiency With a Model of Architecture: From Theory to Practice », *in*: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 37.*10* (2018), pp. 2050–2063, DOI: 10.1109/TCAD.2017.2774822 (cit. on pp. 16, 21, 43, 58, 67–69, 113, 115, 140, 141, 145).

[Per+16] Q. Perret et al., « Temporal Isolation of Hard Real-Time Applications on Many-Core Processors », *in*: *2016 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2016, DOI: 10.1109/RTAS.2016.7461363 (cit. on p. 41).

[Pér+20] A. Pérez et al., « Run-Time Reconfigurable MPSoC-Based On-Board Processor for Vision-Based Space Navigation », *in*: *IEEE Access* 8 (2020), pp. 59891–59905, DOI: 10.1109/ACCESS.2020.2983308 (cit. on pp. 13, 134).

[Pim+02] A. Pimentel et al., « Towards Efficient Design Space Exploration of Heterogeneous Embedded Media Systems. », *in*: *Deprettere E.F., Teich J., Vassiliadis S. (eds) Embedded Processor Design Challenges. SAMOS 2001*, vol. 2268, 2002, pp. 57–73, DOI: 10.1007/3-540-45874-3_4 (cit. on p. 16).

[Pim17] A. D. Pimentel, « Exploring Exploration: A Tutorial Introduction to Embedded Systems Design Space Exploration », *in*: *IEEE Design Test* 34.*1* (2017), pp. 77–90, DOI: 10.1109/MDAT.2016.2626445 (cit. on pp. 21, 22, 136).

[PL95] T. M. Parks and E. A. Lee, « Non-preemptive real-time scheduling of dataflow systems », *in*: *1995 International Conference on Acoustics, Speech, and Signal Processing*, 1995, DOI: 10.1109/ICASSP.1995.479574 (cit. on pp. 21, 137).

[Pli+08] W. Plishker et al., « Functional DIF for Rapid Prototyping », *in*: *2008 The 19th IEEE/IFIP International Symposium on Rapid System Prototyping*, 2008, DOI: 10.1109/RSP.2008.32 (cit. on p. 25).

[PSB09] W. Plishker, N. Sane, and S. S. Bhattacharyya, « A generalized scheduling approach for dynamic dataflow applications », *in*: *2009 Design, Automation & Test in Europe Conference & Exhibition*, 2009, DOI: 10.1109/DATE.2009.5090642 (cit. on p. 25).

[PSR15] F. Palumbo, C. Sau, and L. Raffo, « Coarse-grained reconfiguration: dataflow-based power management », *in*: *IET Computers & Digital Techniques* 9.*1* (2015), pp. 36–48, DOI: 10.1049/iet-cdt.2014.0089 (cit. on pp. 33, 46).

[Pto14] C. Ptolemaeus, *System Design, Modeling, and Simulation using Ptolemy II*, Ptolemy.org, 2014 (cit. on p. 26).

[QP16] W. Quan and A. Pimentel, « A hierarchical run-time adaptive resource allocation framework for large-scale MPSoC systems », *in*: *Design Automation for Embedded Systems* 20.*4* (2016), pp. 311–339, DOI: 10.1007/s10617-016-9179-z (cit. on p. 66).

[Raf+10] E. Raffin et al., « Scheduling, binding and routing system for a run-time reconfigurable operator based multimedia architecture », *in*: *2010 Conference on Design and Architectures for Signal and Image Processing (DASIP)*, 2010, DOI: 10.1109/DASIP.2010.5706261 (cit. on pp. 32, 34, 56, 73).

[Raj+10] R. Rajkumar et al., « Cyber-physical systems: The next computing revolution », *in*: *Design Automation Conference*, 2010, DOI: 10.1145/1837274.1837461 (cit. on pp. 13, 133).

[RBK16] M. Robson, R. Buch, and L. Kalé, « Runtime Coordinated Heterogeneous Tasks in Charm++ », *in*: *2016 Second International Workshop on Extreme Scale Programming Models and Middleware (ESPM2)*, 2016, DOI: 10.1109/ESPM2.2016.011 (cit. on p. 66).

[RDP17] B. Rouxel, S. Derrien, and I. Puaut, « Tightening Contention Delays While Scheduling Parallel Applications on Multi-Core Architectures », *in*: *ACM Transactions on Embedded Computing Systems* 16.*5s* (2017), DOI: 10.1145/3126496 (cit. on p. 23).

[Res] G. V. Research, *Fpga market*, URL: https://www.grandviewresearch.com/industry-analysis/fpga-market (cit. on pp. 29, 137).

[Rez+19] S. H. Rezatofighi et al., « Generalized Intersection Over Union: A Metric and a Loss for Bounding Box Regression », *in*: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2019) (cit. on pp. 64, 145).

[Rih+15] H. Rihani et al., « WCET Analysis in Shared Resources Real-Time Systems with TDMA Buses », *in*: *Proceedings of the 23rd International Conference on Real Time and Networks Systems*, 2015, DOI: 10.1145/2834848.2834871 (cit. on p. 40).

[Ros+07] J. Rosen et al., « Bus Access Optimization for Predictable Implementation of Real-Time Applications on Multiprocessor Systems-on-Chip », *in*: *28th IEEE International Real-Time Systems Symposium (RTSS 2007)*, 2007, DOI: 10.1109/RTSS.2007.24 (cit. on p. 41).

[Sau+14] C. Sau et al., « Automated design flow for coarse-grained reconfigurable platforms: An RVC-CAL multi-standard decoder use-case », *in*: *Proceedings - International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation, SAMOS 2014*, 2014, DOI: 10.1109/SAMOS.2014.6893195 (cit. on pp. 33, 34, 56).

[Sau+15a] C. Sau et al., « Reconfigurable coprocessors synthesis in the MPEG-RVC domain », *in*: *2015 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, 2015, DOI: 10.1109/ReConFig.2015.7393351 (cit. on p. 45).

[Sau+15b] C. Sau et al., « Reconfigurable coprocessors synthesis in the MPEG-RVC domain », *in*: *International Conference on ReConFigurable Computing and FPGAs, ReConFig 2015*, 2015, DOI: 10.1109/ReConFig.2015.7393351 (cit. on p. 46).

[Sau+17] C. Sau et al., « Challenging the Best HEVC Fractional Pixel FPGA Interpolators With Reconfigurable and Multifrequency Approximate Computing », *in*: *IEEE Embedded Systems Letters* 9.*3* (2017), DOI: 10.1109/LES.2017.2703585 (cit. on p. 75).

[SBA13] J. Sérot, F. Berry, and S. Ahmed, « CAPH: a language for implementing stream-processing applications on FPGAs », *in*: *Embedded Systems Design with FPGAs*, Springer, 2013, DOI: 10.1007/978-1-4614-1362-2_9 (cit. on pp. 33, 34, 44, 56, 141).

[Sch+11] A. Schranzhofer et al., « Timing Analysis for Resource Access Interference on Adaptive Resource Arbiters », *in*: *2011 17th IEEE Real-Time and Embedded Technology and Applications Symposium*, 2011, DOI: 10.1109/RTAS.2011.28 (cit. on p. 41).

[SE11] S. Schliecker and R. Ernst, « Real-Time Performance Analysis of Multiprocessor Systems with Shared Memory », *in*: *ACM Transactions on Embedded Computing Systems* 10.*2* (2011), DOI: 10.1145/1880050.1880058 (cit. on p. 41).

[SGB06] S. Stuijk, M. Geilen, and T. Basten, « SDF^3: SDF For Free », *in*: *Application of Concurrency to System Design, 6th International Conference, ACSD 2006, Proceedings*, 2006, DOI: 10.1109/ACSD.2006.23 (cit. on p. 26).

[Sha+14] M. Shafique et al., « Dark Silicon as a Challenge for Hardware/Software Co-Design: Invited Special Session Paper », *in*: *Proceedings of the 2014 International Conference on Hardware/Software Codesign and System Synthesis*, 2014, DOI: 10.1145/2656075.2661645 (cit. on pp. 13, 134).

[SI18] L. R. Still and L. S. Indrusiak, « Memory-Aware Genetic Algorithms for Task Mapping on Hard Real-Time Networks-on-Chip », *in*: *2018 26th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)*, 2018, DOI: 10.1109/PDP2018.2018.00101 (cit. on p. 42).

[Sin+13] A. Singh et al., « Mapping on multi/many-core systems: Survey of current and emerging trends », *in*: *2013 50th ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2013, DOI: 10.1145/2463209.2488734 (cit. on p. 61).

[Sin+17] A. K. Singh et al., « A Survey and Comparative Study of Hard and Soft Real-Time Dynamic Resource Allocation Strategies for Multi-/Many-Core Systems », *in*: *ACM Computing Surveys* 50.*2* (2017), 24:1–24:40, DOI: 10.1145/3057267 (cit. on p. 111).

[Sin07] O. Sinnen, « Task Scheduling for Parallel Systems », *in*: *Wiley series on parallel and distributed computing*, 2007 (cit. on p. 60).

[Sir+10] N. Siret et al., « A codesign synthesis from an MPEG-4 decoder dataflow description », *in*: *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*, 2010, DOI: 10.1109/ISCAS.2010.5537107 (cit. on pp. 32, 56).

[Sma+14] G. Smaragdos et al., « A Dependable Coarse-Grain Reconfigurable Multicore Array », *in*: *Parallel Distributed Processing Symposium Workshops (IPDPSW), 2014 IEEE International*, 2014, DOI: 10.1109/IPDPSW.2014.20 (cit. on p. 30).

[Sou+05] C. C. d. Souza et al., « The Datapath Merging Problem in Reconfigurable Systems: Complexity, Dual Bounds and Heuristic Evaluation », *in*: *ACM Journal of Experimental Algorithmics* 10 (2005), DOI: 10.1145/1064546.1180613 (cit. on p. 45).

[SS17] S. Skalistis and A. Simalatsar, « Near-optimal deployment of dataflow applications on many-core platforms with real-time guarantees », *in*: *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*, 2017, DOI: 10.23919/DATE.2017.7927090 (cit. on p. 42).

[Ste+04] T. Stefanov et al., « System design using Kahn process networks: the Compaan/Laura approach », *in*: *Proceedings of the Design, Automation and Test*

in Europe Conference and Exhibition, 2004, DOI: 10.1109/DATE.2004.1268870 (cit. on pp. 32, 34).

[Syn] Synflow, *Synflow Studio*, URL: www.synflow.com (cit. on pp. 33, 34, 56).

[TB01] R. Tessier and W. Burleson, « Reconfigurable Computing for Digital Signal Processing: A Survey », *in*: *Journal of VLSI Signal Processing-Systems for Signal, Image, and Video Technology* 28.*1-2* (2001), pp. 7–27, DOI: 10.1023/A:1008155020711 (cit. on pp. 29, 137).

[TB04] A. Thomas and J. Becker, « Dynamic Adaptive Runtime Routing Techniques in Multigrain Reconfigurable Hardware Architectures », *in*: *Field Programmable Logic and Application*, ed. by J. Becker, M. Platzner, and S. Vernalde, 2004, pp. 115–124, DOI: 10.1007/978-3-540-30117-2_14 (cit. on p. 66).

[The+06] B. D. Theelen et al., « A scenario-aware data flow model for combined long-run average and worst-case performance analysis », *in*: *Fourth ACM and IEEE International Conference on Formal Methods and Models for Co-Design, 2006. MEMOCODE '06. Proceedings.* 2006, DOI: 10.1109/MEMCOD.2006.1695924 (cit. on p. 24).

[Vor+13] N. Voros et al., « MORPHEUS: A heterogeneous dynamically reconfigurable platform for designing highly complex embedded systems », *in*: *ACM Transactions on Embedded Computing Systems* 12.*3* (2013), DOI: 10.1145/2442116.2442120 (cit. on pp. 32, 34, 56).

[Wen+08] I. Wenzel et al., « Measurement-Based Timing Analysis », *in*: vol. 17, 2008, pp. 430–444, DOI: 10.1007/978-3-540-88479-8_30 (cit. on p. 36).

[Wil+08] R. Wilhelm et al., « The Worst-Case Execution-Time Problem—Overview of Methods and Survey of Tools », *in*: *ACM Transactions on Embedded Computing Systems* 7.*3* (2008), DOI: 10.1145/1347375.1347389 (cit. on pp. 15, 35, 134).

[Wol14] M. Wolf, *High-Performance Embedded Computing, Second Edition: Applications in Cyber-Physical Systems and Mobile Computing*, 2nd, 2014 (cit. on p. 66).

[WP14] S. Wasly and R. Pellizzoni, « Hiding memory latency using fixed priority scheduling », *in*: *2014 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2014, DOI: 10.1109/RTAS.2014.6925992 (cit. on p. 40).

[WWC16] M. Wijtvliet, L. Waeijen, and H. Corporaal, « Coarse grained reconfigurable architectures in the past 25 years: Overview and classification », *in*: *2016 International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation (SAMOS)*, 2016, DOI: 10.1109/SAMOS.2016.7818353 (cit. on pp. 32, 52).

[XAP17] J. Xiao, S. Altmeyer, and A. Pimentel, « Schedulability Analysis of Non-preemptive Real-Time Scheduling for Multicore Processors with Shared Caches », *in*: *2017 IEEE Real-Time Systems Symposium (RTSS)*, 2017, DOI: 10.1109/RTSS.2017.00026 (cit. on p. 41).

[Xil] Xilinx, *Xilinx Vivado High-Level Synthesis*, URL: https://www.xilinx.com/products/design-tools/vivado/integration/esl-design (cit. on pp. 33, 34, 52, 56, 57).

[Xu+16] M. Xu et al., « Analysis and Implementation of Global Preemptive Fixed-Priority Scheduling with Dynamic Cache Allocation », *in*: *2016 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2016, DOI: 10.1109/RTAS.2016.7461322 (cit. on p. 42).

[Yao+12] G. Yao et al., « Memory-centric scheduling for multicore hard real-time systems », *in*: *Real-Time Systems* 48 (2012), pp. 681–715, DOI: 10.1007/s11241-012-9158-9 (cit. on p. 40).

[YKS11] M. Yoon, J. Kim, and L. Sha, « Optimizing Tunable WCET with Shared Resource Allocation and Arbitration in Hard Real-Time Multicore Systems », *in*: *2011 IEEE 32nd Real-Time Systems Symposium*, 2011, DOI: 10.1109/RTSS.2011.28 (cit. on p. 41).

[YPB15] J. Yun, J. Park, and W. Baek, « HARS: A heterogeneity-aware runtime system for self-adaptive multithreaded applications », *in*: *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2015, DOI: 10.1145/2744769.2744848 (cit. on p. 66).

[YPV15] H. Yun, R. Pellizzon, and P. K. Valsan, « Parallelism-Aware Memory Interference Delay Analysis for COTS Multicore Systems », *in*: *2015 27th Euromicro Conference on Real-Time Systems*, 2015, DOI: 10.1109/ECRTS.2015.24 (cit. on p. 40).

[YR93] A. Yeung and J. Rabaey, « A reconfigurable data-driven multiprocessor architecture for rapid prototyping of high throughput DSP algorithms », *in*: *System Sciences, 1993, Proceeding of the Twenty-Sixth Hawaii International Conference on*, 1993, DOI: 10.1109/HICSS.1993.270747 (cit. on p. 31).

[Zha+15] F. Zhang et al., « A task-level adaptive MapReduce framework for real-time streaming data in healthcare applications », *in*: *Future Generation Computer Systems* 43-44 (2015), pp. 149–160, DOI: 10.1016/j.future.2014.06.009 (cit. on p. 66).

[ZWN17] W. Zheng, H. Wu, and C. Nie, « Integrating task scheduling and cache locking for multicore real-time embedded systems », *in*: *ACM SIGPLAN Notices* 52 (2017), pp. 71–80, DOI: 10.1145/3140582.3081033 (cit. on p. 42).

INSA de RENNES
Service des Formations

AVIS DU JURY SUR LA REPRODUCTION DE LA THESE SOUTENUE

Titre de la thèse:
Analyse du temps de réponse des applications de flux de données paramétrées sur des systèmes logiciels/matériels hétérogènes

Nom Prénom de l'auteur : RUBATTU CLAUDIO

Membres du jury :
- Monsieur GAMATIE Abdoulaye
- Monsieur MANCINI Stéphane
- Monsieur CASTRILLON Jeronimo
- Monsieur JUAREZ MARTINEZ Eduardo
- Madame PALUMBO Francesca
- Monsieur RAFFO Luigi
- Monsieur PELCAT Maxime
- Monsieur BHATTACHARYYA Shuvra

Président du jury : RAFFO LUIG.

Date de la soutenance : 02 Décembre 2020

Reproduction de la these soutenue

☒ Thèse pouvant être reproduite en l'état
☐ Thèse pouvant être reproduite après corrections suggérées

Fait à Rennes, le 02 Décembre 2020

Signature du président de jury

Le Directeur,

M'hamed DRISSI

DOCTORAT BRETAGNE LOIRE / MATHSTIC

INSA RENNES

Titre : Analyse du temps de réponse des applications de flux de données paramétrées sur des systèmes logiciels/matériels hétérogènes

Mot clés : Analyse de performance, MPSoC, Flux de données, Synthèse de haut niveau, Reconfiguration à gros grains

Résumé : Les fortes contraintes de réactivité et de consommation énergétique des systèmes embarqués et cyber-physiques nécessitent l'utilisation croissante de systèmes de calculs parallèles et fortement hétérogènes. La nature de ces systèmes parallèles implique une énorme complexité dans la compréhension et la prévision des performances en termes de temps de réponse. En effet, le temps de réponse dépend de nombreux facteurs associés aux caractéristiques à la fois de la fonctionnalité implémentée et de l'architecture cible.

Les méthodes d'optimisation système actuelles dérivent le temps de réponse du système en examinant les opérations requises par chaque tâche, tant pour le traitement que pour l'accès aux ressources partagées. Cette procédure est souvent suivie par l'ajout ou l'élimination des interférences potentielles dues à la concurrence entre tâches. Cependant, de telles approches nécessitent une connaissance avancée des détails du logiciel et du matériel, rarement disponible en pratique lors du dimensionnement du système.

Cette thèse propose une stratégie alternative "top-down" visant à étendre les cas dans lesquels le temps de réponse matériel et logiciel peut être analysé et prédit. La stratégie proposée s'appuie sur des représentations d'applications par des modèles flux de données et se concentre sur l'estimation du temps de réponse d'applications reconfigurables exécutées par des unités de calcul à la fois générales et spécialisées.

Title: Response time analysis of parameterized dataflow applications on heterogeneous SW/HW systems

Keywords: Performance Analysis, MPSoC, Dataflow, HLS, Coarse-Grained Reconfiguration

Abstract: In contexts such as embedded and cyber-physical systems, the design of a desired functionality under constraints increasingly requires a parallel execution of different tasks on heterogeneous architectures. The nature of such parallel systems implies a huge complexity in understanding and predicting performance in terms of response time. Indeed, response time depends on many factors associated with the characteristics of both the functionality and the target architecture.

State-of-the art strategies derive response time by examining the operations required by each task for both processing and accessing shared resources. This procedure is often followed by the addition or elimination of potential interferences due to task concurrency. However, such approaches require an advanced knowledge of the software and hardware details, rarely available in practice.

This thesis provides an alternative "top-down" strategy aimed at extending the cases in which hardware and software response times can be analyzed and predicted. The proposed strategy leverages on dataflow-based application representations and focuses on the response time estimation of reconfigurable applications mapped on both general-purpose and specialized processing elements.