Efficient Adaptive Hard Real-time Multi-processor Systems

THÈSE N^{O} 8146 (2017)

PRÉSENTÉE LE 10 NOVEMBRE 2017
À LA FACULTÉ INFORMATIQUE ET COMMUNICATIONS
LABORATOIRE DES SYSTÈMES INTÉGRÉS (IC/STI)
PROGRAMME DOCTORAL EN INFORMATIQUE ET COMMUNICATIONS

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

POUR L'OBTENTION DU GRADE DE DOCTEUR ÈS SCIENCES

PAR

Stefanos SKALISTIS

acceptée sur proposition du jury:

Prof. J.-Y. Le Boudec, président du jury
Prof. G. De Micheli, Prof. J. Sifakis, directeurs de thèse
Prof. S. Baruah, rapporteur
Prof. I. Puaut, rapporteuse
Prof. D. Atienza Alonso, rapporteur

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Suisse
2017

Οἱ καιροὶ οὐ μϵνϵτοί
— Θουκυδίδης

Times do not wait
— Thucydides

Στους γονείς μου,
Στους διδάσκοντες μου,
Σε όσους πίστεψαν σε εμένα.

To my parents,
To those who taught me,
To those who believed in me.

Acknowledgements

I had the the honour and privilege to be advised by Prof. Joseph Sifakis and Prof. Giovanni De Micheli. As an expert of hard real-time systems once told me: *"No graduate student simply chooses the problem of hard real-time systems; it is always assigned to them by their advisor."* — Prof. Baruah.

I would like to extend my gratitude towards my main advisor, Prof. Sifakis, for providing me with the opportunity of exploring the research problems present in the domain of hard real-time systems. The true value of such systems, as well as the difficulty of the underlying challenges, only becomes apparent after years of efforts, for which I now am thankful. Additionally, his constant persistence towards theoretical and scientific rigour has equipped me with a valuable research skill-set that will accompany me for the rest of my life. I also vastly appreciate Prof. De Micheli's provisions that allowed this work to be completed, as well as his patience and understanding during personally challenging periods. I feel deeply indebted to both of them for this opportunity in my professional life. Their support and guidance is unparalleled and I honestly aspire that some day my contributions will be able to compensate their efforts.

I am grateful to Prof. Sanjoy Baruah, Prof. Isabelle Puaut, Prof. David Atienza Alonso and Prof. Jean-Yves Le Boudec for honouring me in serving as the thesis jury. I thank them for taking the time to read my thesis and for their invaluable comments on my work and its perspectives. I would like to additionally thank Prof. Baruah for being my mentor during a research visit at University of North Carolina at Chapel Hill, which enabled me to significantly increase my knowledge on mixed-criticality systems and scheduling under uncertainty. I thank him for the insightful scientific discussions, which I greatly enjoyed.

I could not possibly omit the unseen heroes behind this dissertation; Dr. Alena Simalatsar and Dr. Federico Angiolini. Their day-to-day efforts in instilling a scientific spirit will never be forgotten. Special thanks go to Dr. Angiolini, Anastasia Mavridou and Stefanos Antaris for proofreading my thesis. Their assistance in improving this manuscript on all levels was precious.

I want to thank all members of the Rigorous System Design (RiSD) lab and Integrated Systems (LSI) lab for creating a pleasant and motivating working environment. I thank Mrs. Arianne Staudenmann and Mrs. Christina Govoni for their constant availability and assistance on administrative tasks, while maintaining a positive spirit at all times.

I would also like to express my gratitude to my dear friends for enriching my life with happiness, joy and laughter. Thank you Abbas, Alex, Androklis, Apostolos, Ashkan,

Despina, Eleni, Eleni, Evangelos, Evgenia, Grigorios, Ignatio, Ioannis, Ioulia, Katerina, Loukia, Manos, Matt, Myrsini, Pavlos, Panagiotis, Stefanos and Theodoros.
Last, but most importantly, I thank my parents, Stamatia and Georgios, for their trust and unconditional love throughout the years. Ultimately, there are no words to express the magnitude of my gratitude towards them. Finally, I am deeply grateful to Georgios Bootsalis; the first man who expressed his belief in me (beyond my family). Without it, I would not have never started this journey.

Lausanne, 10 November 2017 S. S.

Abstract

Modern computing systems are based on *multi-processor* systems, i.e. multiple cores on the same chip. *Hard real-time* systems are required to perform particular tasks within certain amount of time; failure to do so characterises an unaccepted behavior. Hard real-time systems are found in safety-critical applications, e.g. airbag control software, flight control software, etc. In safety-critical applications, failure to meet the real-time constraints can have catastrophic effects.

The *safe* and, at the same time, *efficient* deployment of applications on multi-processors with hard real-time constraints is a challenging task. Scheduling methods and Models of Computation, that provide safe deployments, require a realistic estimation of the Worst-Case Execution Time (WCET) of tasks. The simultaneous access of shared resources by parallel tasks causes *interference delays*, due to hardware arbitration, which affecs WCET. Interference delays can be accounted for, with the pessimistic assumption that all possible interference will occur. Resulting schedules would be exceedingly conservative, thus the benefits of a multi-processor would be significantly negated. Producing less pessimistic schedules is challenging due to the inter-dependency between WCET estimation and deployment optimisation. Accurate estimation of interference delays -and thus estimation of task WCET- depends on the way an application is deployed; deployment is an optimisation problem that depends on the estimation of task WCET. Another efficiency gap, which is of consequence in several systems (e.g. airbag control), stems from the fact that rarely tasks execute with their WCET. Safe runtime adaptation based on the Actual Execution Times, can yield additional improvements in terms of latency (more responsive systems).

To achieve *efficiency* and retain *adaptability*, we propose that interference analysis should be coupled with the deployment process. The proposed interference analysis method estimates the possible amount of interference, based on an architecture and an application model. As more information is provided, such as *scheduling*, *memory mapping*, etc, the per-task interference estimation becomes more accurate. Thus, the method computes *interference-sensitive* WCET estimations (*is*WCET).

Based on the *is*WCET method, we propose a method to break the inter-dependency between WCET estimation and deployment optimisation. Initially, the *is*WCETs are over-approximated, by assuming worst-case interference, and a safe deployment is derived. Subsequently, the proposed method computes accurate *is*WCETs by *spatio-temporal* exclusion, i.e. excluding interferences from non-overlapping tasks that share resources

(space). Based on accurate *is*WCETs, the deployment solution is improved to provide better latency guarantees.
We also propose a distributed runtime adaptation technique, that aims to improve run-time latency. Using *is*WCET estimations restricts the possible adaptations, as an adaptation might increase the interference and violate the safety guarantees. The proposed technique introduces statically scheduling dependencies between tasks that prevent additional interference. At runtime, a self-timed scheduling policy that respects these dependencies, is applied, proven to be safe, and with minimal overhead.
Experimental evaluation on Kalray MPPA-256 shows that our methods improve *is*WCET up to 36%, guaranteed latency up to 46%, runtime performance up to 42%, with a consolidated performance gain of 50%.

Key words: hard real-time systems, interference-sensitive WCET, runtime adaptation, Kalray MPPA-256

Résumé

Les systèmes informatiques modernes sont basés sur des systèmes multiprocesseurs, c'est-à-dire avec plusieurs cœurs sur la même puce. Des systèmes temps-réel durs sont nécessaires pour effectuer des tâches particulières en un certain laps de temps ; un échec caractérise un comportement non accepté. Des systèmes temps-réel durs se retrouvent dans des applications critiques pour la sécurité, par exemple dans les logiciels de contrôle des airbags, les logiciels de contrôle de vol, etc.. Pour des applications critiques pour la sécurité, le non-respect des contraintes de temps réel peut avoir des effets catastrophiques.

Le déploiement *sûr* et, en même temps, *efficace* des applications sur un multiprocesseur, sous des contraintes de temps réel dur, pose un problème difficile. Les méthodes d'ordonnancement et les modèles de calcul visées à des déploiements sûrs nécessitent une estimation réaliste du temps d'exécution le plus pessimiste (Worst Case Execution Time - WCET) des tâches. L'accès simultané par des tâches parallèles aux ressources partagées entraîne des *retards d'interférence* en raison de l'arbitrage hardware. Les retards d'interférence peuvent bien être pris en compte, en faisant l'hypothèse pessimiste selon laquelle toute interférence possible se produirait. Les ordonnancements résultants seraient extrêmement conservateurs, de sorte que les avantages du multiprocesseur seraient significativement réduits. Cependant, produire des ordonnancements moins pessimistes est difficile en raison de l'interdépendance entre l'estimation du WCET et l'optimisation du déploiement. L'estimation précise des retards d'interférence - et donc l'estimation du WCET des tâches - dépend de la façon dont une application est déployée ; le déploiement est un problème d'optimisation qui dépend de l'estimation du WCET des tâches. Une autre inefficacité, qui peut être lourde dans plusieurs systèmes (par exemple, le contrôle des airbags), découle du fait que rarement les tâches s'exécutent avec leur WCET. Une adaptation de l'exécution basée sur les temps d'exécution réels, pourvu qu'elle soit sûre, peut apporter des améliorations supplémentaires en termes de temps de réponse (systèmes plus réactifs).

Pour atteindre l'*efficacité* et conserver l'*adaptabilité*, nous proposons que l'analyse des interférences soit associée au processus de déploiement. La méthode d'analyse d'interférence proposée identifie les interférences possibles, en fonction d'une architecture et d'un modèle d'application. À mesure que d'autres informations sont fournies, telles que l'ordonnancement, la répartition de la mémoire, etc., l'estimation de l'interférence pour chaque tâche devient plus précise. Ainsi, la méthode calcule un temps d'exécution tenant les interférences en compte (*interference-sensitive* WCET - *is*WCET).

Sur la base de l'*is*WCET, nous proposons une méthode pour découpler l'interdépendance entre l'estimation du WCET et l'optimisation du déploiement. En une première phase, les *is*WCET sont surestimés, en supposant les interférences les plus défavorables, et un déploiement sûr est dérivé. Par la suite, la méthode proposée calcule des *is*WCETs plus précis par exclusion spatio-temporelle, c'est-à-dire en excluant les interférences des tâches qui partagent des ressources (espace) mais ne se chevauchent pas dans le temps. Basé sur ces *is*WCETs précis, la solution de déploiement est améliorée pour optimiser les temps de réponse.
Nous proposons également une technique d'adaptation répartie, qui vise à améliorer les temps de réponse en temps réel. L'utilisation des estimations d'*is*WCET restreint les adaptations possibles, car une adaptation pourrait augmenter les interférences et violer les garanties de sûreté. La technique proposée introduit statiquement des dépendances d'ordonnancement entre les tâches, ce qui empêche l'apparition d'interférences supplémentaires. Au moment de l'exécution, une politique d'ordonnancement auto-cadencée qui respecte ces dépendances est appliquée, et prouvée comme étant sûre et peu lourde.
L'évaluation expérimentale sur une plate-forme Kalray MPPA-256 montre que nos méthodes améliorent les isWCETs jusqu'à 36%, les temps de réponse garantis jusqu'à 46%, et les performances jusqu'à 42%, avec un gain de performance consolidé de 50%.

Mots clefs : systèmes en temps réel durs, WCET sensible aux interférences, adaptation d'exécution, Kalray MPPA-256, NoC

Contents

Acknowledgements **i**

Abstract (English/Français) **iii**

List of figures **ix**

List of tables **xi**

1 Introduction **1**

1.1 The need of Multi-processors 1

1.2 Challenges in Hard Real-time Multi-processor Systems 2

1.3 Thesis Statement and Contributions 7

1.4 Dissertation Outline 8

2 Preliminaries **11**

2.1 Real-time Deployment Problem Formulation 14

2.2 Processor Architecture Models 15

2.2.1 Homogeneous Architecture Types 16

2.2.2 Generic Architecture Model 18

2.2.3 Kalray MPPA-256 19

2.3 Application and System model 20

2.3.1 Application Model 23

2.3.2 System Model 25

2.3.3 Deployment 26

3 Interference-sensitive WCET **29**

3.1 Interference Model 30

3.1.1 Hardware Arbitration Policies 32

3.2 Interference Analysis 34

3.2.1 Intra-cluster Interference 37

3.2.2 Inter-cluster Interference 38

3.2.3 NoC Interference 39

3.2.4 Safety 41

3.3 Evaluation 43

3.3.1 Experimental Setup 43
3.3.2 Evaluation Results 44
3.4 Related work 48
3.4.1 WCET Analysis and Scheduling 49
3.5 Summary 51

4 Application Deployment **53**
4.1 Safe Deployment 55
4.1.1 Partitioning/Placement 56
4.1.2 Acquiring the System Model 57
4.1.3 Scheduling, Task Mapping and FIFO Allocation 60
4.1.4 Memory Mapping, NoC Routing 62
4.1.5 Deployment tightening 63
4.1.6 Safety 64
4.2 Evaluation 66
4.2.1 Experimental Setup 66
4.2.2 Evaluation Results 66
4.3 Related work 68
4.4 Summary 70

5 Runtime Adaptation **73**
5.1 Theoretical Results 75
5.2 Interference-sensitive Runtime Adaptation 78
5.2.1 Runtime Monitor Operation 80
5.2.2 Monitor Overhead 81
5.2.3 Safety 82
5.3 Evaluation 83
5.3.1 Evaluation Setup 83
5.3.2 Evaluation Results 84
5.4 Related work 86
5.5 Summary 87

6 Conclusions and Future Work **89**

Bibliography **102**

Curriculum Vitae **103**

List of Figures

1.1 Power trends for desktop processors 2
1.2 A typical design process for hard real-time uni-processor systems 3
1.3 Example of varying WCET due to interference 4
1.4 “WCET estimation” and “deployment optimisation” inter-dependency . . 5
1.5 State-of-the-art 6
1.6 Global overview of the proposed approaches 8

2.1 Abstract and concrete models 13
2.2 Optimal real-time deployment for known WCET 14
2.3 Different types of processor architectures. 15
2.4 A typical centralised multi-processor architecture. 16
2.5 A typical distributed multi-processor architecture. 17
2.6 A typical mixed multi-processor architecture. 18
2.7 Kalray MPPA-256 compute cluster 19
2.8 Kalray MPPA-256 NoC topology 20
2.9 Example of underflow protection 23
2.10 A split join graph 24
2.11 Corresponding task graph, with arrows denoting the input/output FIFOs 24
2.12 Example of a system model 26

3.1 Overview of the *is*WCET approach 30
3.2 Kalray MPPA-256 memory bank arbitration 39
3.3 Kalray MPPA-256 NoC router arbitration 40
3.4 Percentage of spatio-temporally excluded interference 45
3.5 Percentage overhead due to interference 46
3.6 Task model for various data-parallelisation factors 47
3.7 Latency reduction due to data-parallelisation 47
3.8 Average percentage reduction of tasks WCET 48

4.1 Overview of the deployment approach. 54
4.2 System model for two dependent actors 59
4.3 Example of schedules β and β' 65
4.7 Percentage of guaranteed latency reduction 68

5.1 Example task set that can cause a timing-anomaly 76
5.2 Example of a timing-anomaly 76
5.3 Example of monitor operation for four tasks on two cores 80
5.4 Guaranteed (offline) and runtime (online) latency gain 85
5.5 Comparison of the observed latency vs the guaranteed latency of the produce-process-consume application 86

6.1 Deployment process, revised for multi-processors. 90

List of Tables

3.1 Projections of a task-set V, a dependency relation E and an execution model $EM = (V, E)$ 37
3.2 Kalray MPPA-256 arbitration constants 43
3.3 Worst-case latency of StreamIt benchmarks deployed on Kalray MPPA-256 44

4.1 Extension of functions $\alpha, \delta_{iso}, \sigma$ for communication actors 59

5.1 Monitor computation overhead 82
5.2 Monitor interference overhead 82

1 Introduction

1.1 The need of Multi-processors

Modern computing systems are based on *multi-processor* systems, i.e. processors with more than one processing element (PE) integrated in the same chip. Architectures based on the multi-processor paradigm allowed systems to scale and accommodate more demanding applications. This is apparent nowadays, from large distributed systems and cloud services to mobile applications and autonomous vehicles. The main reason that led the processor manufacturing industry to the multi-processor paradigm was the need for higher performance. While it is not the purpose of this thesis to study the concrete phenomena that mandated the use of multi-processors, it is important to outline the main reasons, as they do affect hard real-time systems which did not scale as the rest computing systems. From the '80s until the beginning of the millennium, the processor manufacturing industry was focusing its efforts in increasing the density of transistors in a chip. Smaller transistors present smaller capacitive loads to their drivers, which can thus switch faster. As an immediate consequence, transistor-based units could be operated at higher clock frequencies, resulting in faster processors. This would lead in faster processors that could meet the increasing processing demand.

Despite that the increase of transistor density continued at the same rate for years, known as Moore's Law [92], the processor manufacturing industry shifted its focus to *multi-processor* architectures. The main reasons for this shift are related to power demand, heat dissipation and operating frequency. As a higher operating frequency requires a higher power supply, it results in higher heat dissipation due to power leakage. Therefore, there is limit in frequency at which a chip can be operated for long periods of time without overheating and requiring a reasonable amount of power [6]. This limit, often called Power Wall [65], was reached in 2004 when prototypes of the Tejas processor (the cancelled Pentium IV successor), operating at 2.8 GHz, required 150 Watts.

The proposed alternative to accommodate the demand for more processing power was

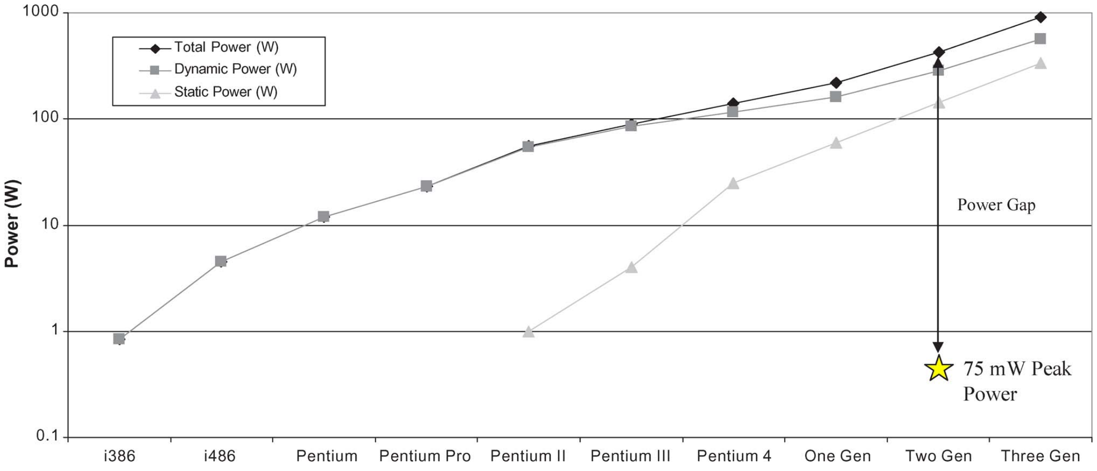


Figure 1.1 – Trends of required power for desktop processors [113]

multi-processor architectures, where multiple PEs — called *cores* — are bundled in the same integrated circuit. The first designs and implementations consisted of 8 or fewer cores on the same die, sharing resources of the chip, e.g. memories, bus interconnect, buses to the shared and main memory, etc. This scheme could not scale up to several dozens or hundreds of cores, as sharing the same communication medium was deemed an impractical approach [55]. This gave birth to a new interconnection paradigm, the *Network-on-Chip* (NoC) [12, 32], which consists of routing nodes, network interfaces and links among them forming a network. Behind each network interface are located one (or more) cores with a private memory. Cores can exchange data over the NoC, accessed through the network interface, thus decoupling computation and communication operations. NoC routers, as in computer networks, are responsible for data routing between the source and target core.

1.2 Challenges in Hard Real-time Multi-processor Systems

While multi-processor and NoC based architectures were realised to meet the increasing processing demand in several computing domains, the fact that multiple resources are extensively shared, compared to uniprocessor systems, has a significant impact for *real-time computing*. Real-time systems are computing systems which are required to perform particular tasks within certain amount of time, called deadlines. Whether there is value in responding beyond the deadline distinguishes *hard* from *soft* real-time systems [26, 14]. In order to prove that deadlines are met, hard real-time systems are required to be formally modelled, while soft real-time – being defined as not hard – have no such requirement, in general. For example, the airbag control system of modern cars is a hard real-time system, as deploying the airbag after its deadline can potentially harm the driver. On the other hand, responding to keyboard strokes or mouse clicks is considered a soft real-time, as it is desirable to react to such events even past their deadlines. Therefore,

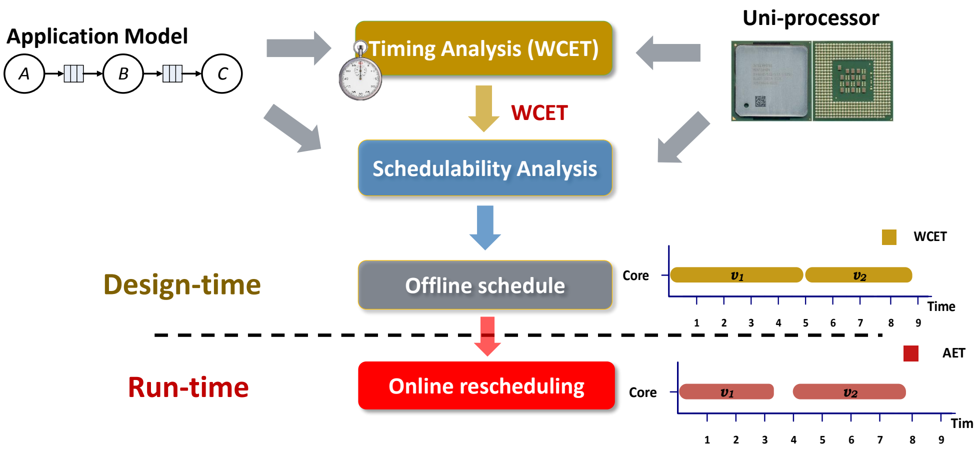


Figure 1.2 – A typical design process for hard real-time uni-processor systems

in order to prove that deadlines are met, it is important for hard real-time systems to have analysable and predictable timing behavior.

Modelling, analysing and optimising the timing behavior of uniprocessor-based real-time systems was the focus for several decades, for the Scheduling [14, 26, 82] and Verification communities [53, 22, 75, 31]. Formal models of real-time systems are based on the notion of *tasks*, which are units of execution, ranging from a bundle of instructions up to a whole set of programming functions. Using such models, the timing behavior of the system can then be studied, analysed and/or optimised. Typically, approaches that provide hard real-time guarantees [14, 26, 34, 25] are based on the notion of *Worst-Case Execution Time* (WCET) of tasks, i.e. the maximum amount of time required for a task to execute in the target architecture under any possible valid input [112, 41]. Acquiring the exact WCET for a task is rather challenging, and not feasible in all cases. Acquiring exact WCET requires to consider all possible execution paths of the application (at source/binary code level) an all possible execution scenarios of instructions at the target architecture, including out-of-order execution, branch prediction, cache replacement policy, data/instruction prefetching, etc. [112, 110]. This leads to an exponential number of possible executions which have to be explored. Even for small-sized applications this exploration cost is prohibiting, rendering acquisition of exact WCET impractical [112].

Figure 1.2 illustrates a typical design process [26, 34, 25, 14, 112, 31] for realizing hard real-time uni-processor systems. Given a set of application tasks, the first step of such a process is to perform timing analysis of those tasks in order to acquire their WCET estimation for the target architecture. Using these WCET estimations, the process performs a schedulability analysis, for a given scheduling policy decided at design-time, to determine if all the real-time constraints (internal deadlines, latency guarantees, etc.) are met. Upon success, the process may generate (offline) static schedules and timing information, which could be used as input for a run-time scheduler. At runtime, the

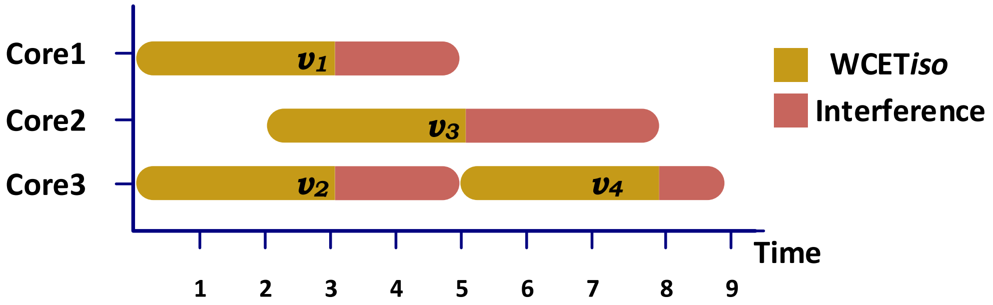


Figure 1.3 – Example of varying WCET for four equivalent tasks executed on three cores. Tasks v_1, v_2 will experience, in the worst-case, two time units of interference due to their mutual overlapping and the overlapping with v_3; Task v_3 will experience three time units from tasks v_1, v_2, v_4, while task v_4 only one time unit from v_3 (WCET*iso* denotes WCET in isolation, i.e. in absence of interference)

predetermined scheduling policy is implemented or, if additional information is provided, a run-time policy is enforced which, nevertheless, must be proven that preserves the acquired real-time guarantees (deadlines, latency).

Certainly, there are several variations of the “typical” design process for realising hard real-time systems, each with its benefits and shortcomings, on a uni-processor. However, with the rise of new architectures with multiple PEs, timing analysis and optimisation becomes even more challenging [14, 33, 105, 70, 40]. This is due to sharing several arbitrated resources among these PEs (memories and interconnects) which introduces timing delays and changes the timing behavior in a non-deterministic manner. As pointed out by Davis and Burns [34]: *“A recent survey of WCET techniques [112] concluded that no static analysis currently exists for multi-core processors which have such*[1] *complex hardware interactions. Research in this area has either aimed to develop a fully statically analysable multi-core CPU, or to move from providing deterministic WCET estimates to providing probabilistic ones with a high degree of confidence.”*

A motivating example of the variability of WCET estimations, due to resource sharing, is outlined in Figure 1.3. Consider four equivalent tasks, e.g. an image filter applied to four different parts of an image. These tasks, being equivalent, have the same WCET when executed in isolation. If the overhead per overlapping task is of one unit, it is clear that the WCET of each task will increase, with the amount of WCET increase being dependant on how and when these tasks are executed. Therefore, despite the fact that these tasks are equivalent, thus have the same WCET in isolation, their WCET varies according to the chosen deployment.

In order to naturally extend the uni-processor deployment approaches to multi-processor architectures, one would have to over-approximate the tasks WCET, such that the WCET

[1] Referring to timing delays due to contention on shared resources

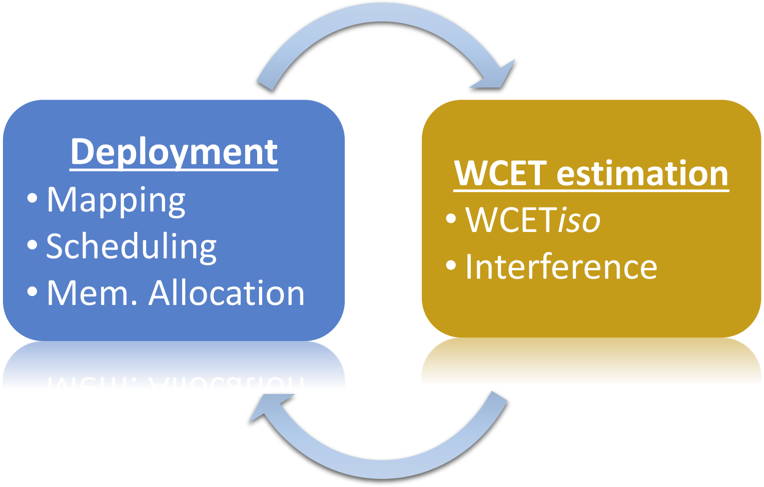
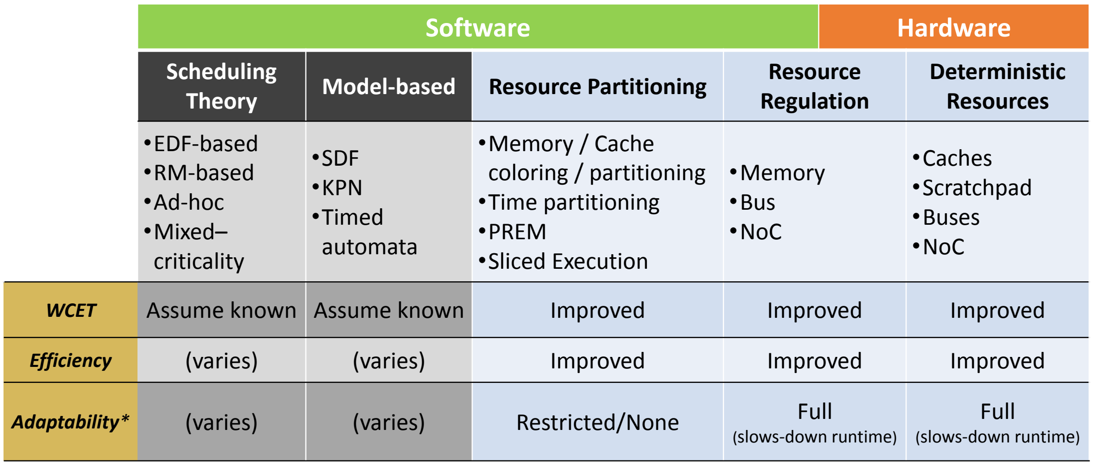


Figure 1.4 – “WCET estimation” and “deployment optimisation” inter-dependency; optimisation of the former depends on the latter, and vice versa

accounts for all possible interferences. Nevertheless, such a pessimistic WCET estimation will result in under-utilised systems, thus seriously undermining the advantages of utilising multi-processors. This is often called the “one-out-of-m processors” problem [63], where the additional processing capacity is negated by the pessimism of the WCET. In fact recent research [64], including research performed within this dissertation [100, 99, 97] shows that WCET estimations which account for interferences from parallel tasks can be 150%, or even 750%, of the corresponding estimations in absence of interference.

Yet, the vast majority of scheduling approaches that aim to address hard real-time systems, either for standard task models [34, 14, 26, 82, 50] or Models of Computation (MoCs) [68, 49, 67, 17, 72, 19], require the *a-priori* knowledge of tasks WCETs. This, of course, is not without reason; one being that research performed from the ’60s until late ’90s made a then reasonable assumption that delays due to arbitration of shared resources are negligible since processors operated in low frequencies and comprised of a low-number of cores.

Another, and more important reason for the *a-priori* knowledge of WCET assumption, is the inter-dependency between “WCET estimation” and “deployment optimisation” that appears in multi-processor systems [34]. Accurate estimation of interference delays -and by extension estimation of tasks WCET- depends on the way an application is deployed. But, optimisation of hard real-time deployment (in terms of latency) is a constrained optimisation problem that depends on the estimation of tasks WCET, thus creating a cyclic dependency between the two main aspects of realising hard real-time systems.



	Software				Hardware
	Scheduling Theory	Model-based	Resource Partitioning	Resource Regulation	Deterministic Resources
	•EDF-based •RM-based •Ad-hoc •Mixed–criticality	•SDF •KPN •Timed automata	•Memory / Cache coloring / partitioning •Time partitioning •PREM •Sliced Execution	•Memory •Bus •NoC	•Caches •Scratchpad •Buses •NoC
WCET	Assume known	Assume known	Improved	Improved	Improved
Efficiency	(varies)	(varies)	Improved	Improved	Improved
Adaptability*	(varies)	(varies)	Restricted/None	Full (slows-down runtime)	Full (slows-down runtime)

* Ratio of safe adaptations to total adaptations (directly affects runtime performance)

Figure 1.5 – Overview of the state-of-art

In order to resolve this cyclic dependencies and still acquire reasonable WCET estimations, several state-of-the-art approaches (Figure 1.5) employ resource isolation [34, 94, 21, 81], or resource regulation [78, 36, 66, 64] techniques. In resource isolation, access to shared resources (buses, memories, NoC routers) is restricted to strictly one core (e.g. spatial partitioning [102], memory coloring) or one core at a given time (e.g. temporal partitioning [23, 48], temporal isolation [21, 81]). This is achieved either at the hardware level (e.g. TDMA) or using software mechanisms [34]. Resource regulation techniques, on the other hand, allow the simultaneous use of resources, but restrict the amount of requests to a shared resource per core within a predefined time window.

While such approaches restrict the pessimism in WCET estimations, and thus provide more *efficient* deployments, they can undermine run-time performance as they restrict *adaptability*. Consider, as an example, the hypothetical scenario where the airbag (SRS) control software of a vehicle is executed in the same multi-processor as the automatic braking (ABS) control software. Both software have real-time guarantees and there is a benefit in executing them as fast as possible. In the case of resource isolation techniques, the SRS cannot access resources dedicated to ABS while ABS is executing, and vice versa. In the case of resource regulation, while accessing a shared resource is permitted, the amount of requests is restricted over time, which essentially slows down the execution of both the SRS and ABS software.

This dissertation holistically provides methods for deployment of applications to multi-processors architectures, as the authors of [33, 104], but with the aim of providing hard real-time guarantees. In order to realise efficient hard real-time systems, without sacrificing adaptability [94, 21, 81], task WCETs are improved as the deployment process progresses. A similar idea [41] is being advocated by the on-going European-funded project Argo [84].

1.3 Thesis Statement and Contributions

This dissertation aims to stand between the two ends of the spectrum of approaches, that is (i) methods that do not restrict *adaptability*, but overestimate WCET and provide pessimistic latency guarantees, and (ii) methods that provide *accurate* WCET estimations and tight latency guarantees, but restrict run-time performance. To bridge that gap, this dissertation proposes novel methods for safely deploying a streaming application to a generic homogeneous architecture, by coupling the WCET estimation method to the deployment process.

The first proposed method is an interference analysis, called *is*WCET method, which given an architecture model, an application model with data-dependencies, and the WCET estimations of the application's tasks (in absence of interference), estimates the possible amount of interference. As more information is provided, such as *task mapping*, *scheduling*, *memory mapping*, etc., the per-task interference estimation becomes more accurate, but also more sensitive to changes. These *interference-sensitive* estimations are considered safe only for deployments in which the maximum interference each task can experience is at most as those interference-sensitive estimations.

The second proposal is to couple such an *is*WCET method to a deployment process for a multi-processor architecture, as shown in Figure 1.6. The proposed process derives safe (in terms or real-time constraints, buffer protection and deadlock freedom) and efficient deployments, by gradually providing information to the *is*WCET method. To illustrate the benefits of such coupling, we apply such an approach to a variant of SDF graphs, called split-join graphs. Intuitively, split-join graphs model explicitly task-/data-parallelism of actors, by allowing potentially parallel firings of actors to read/write to the same channel.

Still, while the generated deployment solution is safe and improved in terms of guaranteed latency, run-time performance can be severely undermined, since the worst-case is unlikely to occur at runtime. While the *is*WCET-based deployments are more efficient, in order to avoid violating the real-time guarantees, such deployments permit only certain adaptations out of the possible many. The adaptations that are deemed safe should not increase the interference any task may suffer in the worst-case. Therefore, the third contribution of this dissertation is a novel, interference-sensitive, runtime adaptation technique, called *is*RA, which is suitable for deployments based on *is*WCET estimations. Being a distributed approach, the execution of the runtime monitors responsible for enforcing the adaptation policy can also cause interference. For this reason, the runtime monitors are designed, and implemented, having minimal execution time and induced interference, thus avoiding undermining runtime performance and still meeting the real-time guarantees.

The final contribution of this thesis, is the realisation of all the aforementioned methods and techniques for an actual multi-processor (Kalray MPPA-256). All of the approaches

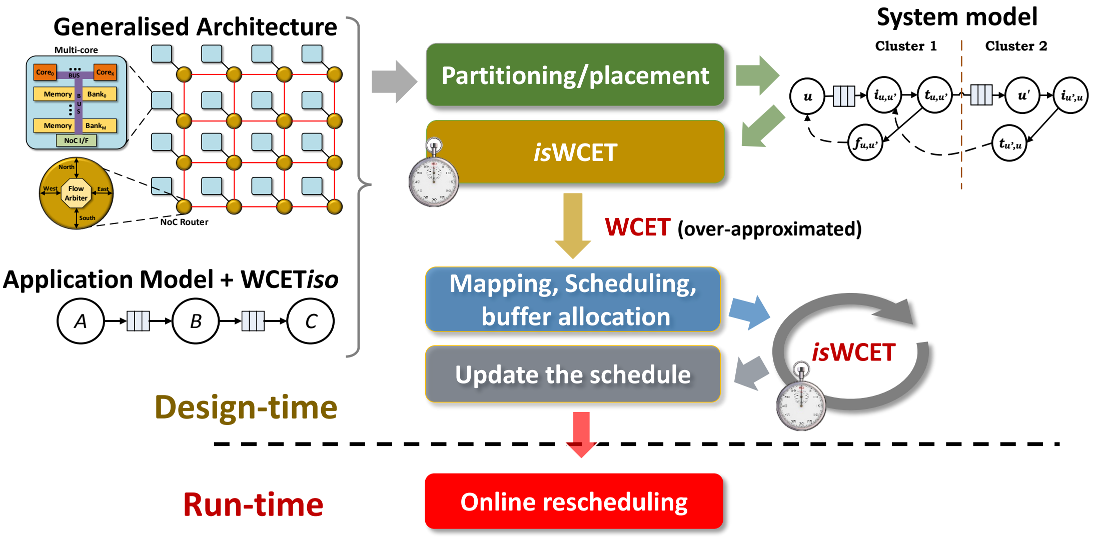


Figure 1.6 – Global overview of the proposed approaches

were evaluated on this multi-processor with real-life applications (StreamIt benchmark), outlining the benefits that each method contributes to the the problem of application deployment with real-time guarantees.

1.4 Dissertation Outline

Chapter 2 presents the fundamental models used for streaming applications and generic architectures. Based upon these models, an execution model is derived which faithfully captures possible executions of the application deployed on a generic architecture and serves as the intermediate representation for the various steps of the proposed approaches.

Based on an execution model, in *Chapter 3*, a novel interference analysis and WCET estimation method is presented, called *is*WCET. As more information is available about the application deployment (such as memory mapping, task mapping, task scheduling, etc.), the *is*WCET method provides more accurate interference estimations, via iteratively applying *spatio-temporal* exclusion, thus acquiring more accurate, but sensitive, WCET estimations. The method is formally proven to be (i) *safe*, (ii) *exact* and (iii) always *converges*. Experimental results of the StreamIt benchmark deployed on the Kalray MPPA-256 multi-processor show that the *is*WCET method is able to exclude 74% of the interference delays on average, improve WCET estimations up to 36% and reduce the average overhead due to interference to less than 10%.

In *Chapter 4*, a safe deployment process is presented that leverages the outcomes of the *is*WCET method. The deployment method manages to break the inter-dependency cycle by coupling the WCET estimation with the deployment process. Specifically, the proposed deployment process estimates tasks WCET using the *is*WCET method, without any information regarding the deployment. Once a safe deployment is acquired, the process provides the deployment information to the *is*WCET method to derive tighter WCET estimations and adapts accordingly the acquired deployment. Extending the results of Chapter 3, experimental evaluation shows that coupling the *is*WCET with the deployment process, can improve guaranteed latency up to 46%.

Chapter 5 presents a novel distributed runtime adaptation technique, called *is*RA, which builds upon *is*WCET estimations and aims at improving run-time latency. The proposed *is*RA technique restricts executions that would introduce additional interferences, beyond the ones already accounted for by the deployment process. The method is proven to be safe and illustrated to have minimal overhead in execution time and induced interference. The *is*RA is experimentally evaluated with the StreamIt benchmark deployed on the Kalray MPPA-256 multi-processor, using the *is*WCET method and the proposed deployment process. Experimental results show that the *is*RA improves runtime latency by 22% on average, up to 42%. Thus, the consolidated improvement in latency, from all the proposed methods, can be of more than 50%.

2 Preliminaries

To provide real-time guarantees for an application deployed on an architecture, formal models for the application and the target architecture are necessary. An application is often considered in literature (e.g. [82, 14, 40, 34]) to consist of a set of *computation tasks* which may be data-dependent. When all these computation tasks are executed, the application is considered to have completed its execution, that is, the input data have been consumed and the desired output has been produced. The output is considered to be correct if during the execution there are no data-races, deadlocks etc. In a very abstract manner, we can represent an application as a *task graph* $A = (V', E')$ where V' is the set of computation tasks and E' is the dependency relation among the computation tasks. Deploying an application on a multi-processor architecture may also require additional tasks in order to faithfully model *architecture-specific* operations of the underlying architecture, such as hardware synchronisation, NoC transfers, etc. Therefore, the resulting system can be viewed as a task graph $S = (V, E)$, derived from the application graph, which contains the original computation and additional architecture-specific tasks and respects the data-dependencies of the application.

Given a set of given real-time constraints for an application, the real-time deployment of an application on a multi-processor architecture involves several decisions. These decisions drill down to well known problems, that is (i) *processor mapping*, (ii) *memory mapping*, (iii) *NoC routing* and (iv) *processor scheduling*. Processor mapping (often also called task mapping) refers to the decision of assigning tasks to PEs (without considering the order of execution), while processor scheduling is the decision of the order of task execution. Often, especially in Scheduling Theory [82], these are collectively referred as scheduling problem, but we will make the aforementioned distinction as it has a significant impact on our approaches. The memory mapping problem refers to the decision of which parts of the memory will be assigned to each task in order to perform its execution. Finally, NoC routing is a typical network problem of deciding the sequence of routers that packets or flows should traverse, in order to reach their destination. All these decisions have an impact on the resource demands and execution times of tasks, thus affecting the efficiency of the system.

As a semi-formal definition of what a *real-time deployment* consists of, let the tuple $D = (\mu_K, \mu_M, \rho, \beta, \epsilon)$ be the non-preemptive *deployment solution*, i.e the collective solution of these problems, where:

- μ_K is the function which assigns tasks to PEs (*processor or task mapping* problem)
- μ_M is the function which maps tasks data-dependencies, private variables and binary code to memory (*memory mapping* problem)
- ρ is the routing of the data-flows on the NoC (*NoC routing* problem)
- β and ϵ are the functions which denote the begin and end times of tasks, respectively (*scheduling* problem)

In order to generate a deployment solution which meets certain real-time constraints, e.g. tasks finish before some given deadlines and/or the execution finishes prior to a given latency constraint, timing information of the application's tasks must be provided. The required timing information for each task is called *Worst-Case Execution Time* (WCET), which is the maximum amount of time required for a task to execute in the target architecture under any possible valid input. In architectures with multiple PEs the actual WCET of any task v can vary according to the *deployment*, due to sharing arbitrated resources among PEs which introduces *interference* delays and changes the timing behavior in a non-deterministic way.

In principle, the WCET of a task v can be decomposed into the WCET in isolation, i.e. in absence of interference, and the interference delays. Thus, given a deployment solution D, for any task v its total duration of execution is:

$$\delta_{\iota^D}(v) \overset{def}{=} \delta_{iso}(v) + \iota^D(v) \tag{2.1}$$

where $\delta_{iso}(v)$ is the WCET in isolation, which is assumed constant regardless of the deployment D, and $\iota^D(v)$ is the amount of delays that task v experiences from simultaneously executed tasks. The assumption that the WCET in isolation is constant, and does not depend on the deployment D, implies a homogeneous architecture, i.e. all PEs are identical. Even in such architectures, while the WCET in isolation $\delta_{iso}(v)$ for any task v does not depend on the deployment D, the same is not true for the interference delays $\iota^D(v)$ as they vary based on the deployment D. We will call δ_{ι^D} the *interference-sensitive* WCET, or simply *is*WCET.

Our approaches involve several models (Figure 2.1); Initially, two abstract models are consider as input, i.e. *application model* and *architecture model*. These are used to construct the *system model*, which faithfully models the execution of the *application*

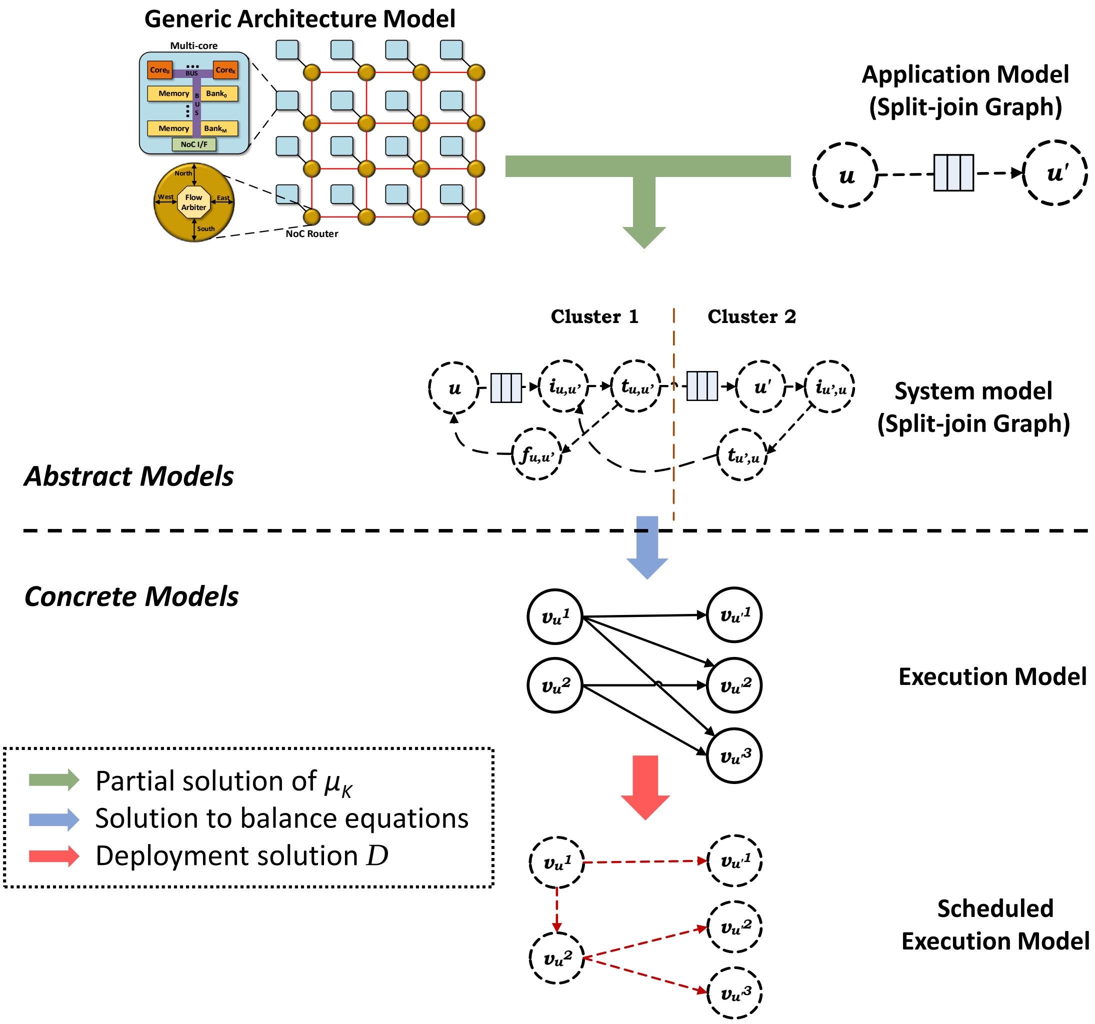


Figure 2.1 – Abstract and concrete models

model to the target *architecture model*. The system model is transformed in the execution model, which contains all the tasks of the system. The purpose is to solve the majority of optimisation problems, required to acquire a safe deployment, on the abstract model, i.e. system model. Whenever the system model cannot be used efficiently, its corresponding execution model is used. The execution model, serves also another purpose. When the safe deployment is derived, the execution model is transformed to the scheduled execution model, which is the input for our runtime adaptation, along with the deployment solution. This transformation of the execution model will play a crucial role in the efficiency of the proposed runtime adaptation, but this will be discussed on Chapter 5.

In the next subsections the real-time deployment problem is formulated and the aforementioned abstract and concrete models (in Figure 2.1) are formally defined.

2.1 Real-time Deployment Problem Formulation

The *optimal*, in terms of time, *deployment* of an application on the target architecture consists of finding the solution with the smallest *latency* to the i) processor mapping (μ_K), ii) memory mapping (μ_M), iii) NoC routing (ρ) and iv) scheduling (β,ϵ) sub-problems, such that given *real-time* constraints are met. Thus, the real-time deployment problem can be formulated as a constrained optimisation problem:

$$\text{Find } D = (\mu_K, \mu_M, \rho, \beta, \epsilon) \ \text{ s.t. } \min\left(\max_{v \in V}(\epsilon(v))\right) \text{ subject to } C_{RT}$$

where C_{RT} is a set of real-time constraints of the form "*some tasks should finish before their deadline*", i.e. $v \in V,\ \epsilon(v) \le d_v$. To illustrate a simplified version of the whole problem, consider Figure 2.2, where four tasks are deployed on two cores. Each task has a (known) WCET of three times units and its respective deadlines is illustrated with a dashed-line.

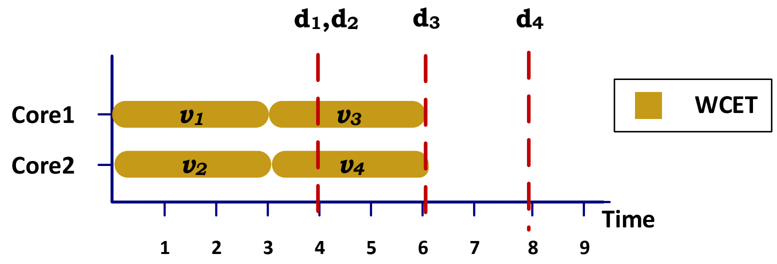


Figure 2.2 – Optimal real-time deployment for known WCET

The focus of this dissertation is on *non-preemptive, homogeneous* systems. In homogeneous systems the PEs of the architecture are identical and the WCET in isolation, $\delta_{iso}(v)$, of any task $v \in V$ is the same on all PEs. Non-preemptive systems are those in which once a task has started its execution, it cannot be interrupted and the task's required resources (memory locations, PE) cannot be used until the task finishes execution. Thus, given the *is*WCET δ_{ι_D} the task-end function ϵ, for non-preemptive systems, is defined as:

$$\epsilon(v) \overset{def}{=} \beta(v) + \delta_{\iota_D}(v) \quad \forall v \in V \tag{2.2}$$

Notice that even for non-preemptive systems the optimisation problem becomes rather complicated, as the exact valuation of the optimisation function ϵ depends on the solution of the optimisation problem D.

There are several variations of the same problem depending on the type of architecture and the assumptions made about the WCET. For example, for a multi-processor architecture with a bus-based interconnect, the deployment reduces to a tuple without the NoC routing function ρ (i.e. $D = (\mu_K, \mu_M, \beta)$), while if the WCET estimations are pessimistically estimated to include all possible interference delays, the deployment can even reduce to $D = (\mu_K, \beta)$. These variations can reduce the complexity of the deployment problem but also lead to inefficient systems (details can be found in Section 3). As opposed to many techniques in the literature, this dissertation does not rely on overly pessimistic WCET estimations, since one of its aims is to explore and outline the impact of such pessimism to the efficiency of the resulting system.

2.2 Processor Architecture Models

At the time this dissertation was written, several research [58] and commercial [61, 108, 60, 87, 46] computing platforms with multiple PEs were available. These broadly can be categorised according to i) the type of PEs and ii) the on-chip memory organisation, as illustrated in Figure 2.3.

Homogeneous (as opposed to *heterogeneous*) are the platforms where the PEs are of the same type, e.g. CPU [60, 80], GPU [79], DSP, and have the same overall speed when executing a task, i.e. all PEs have the same clock speed, cache size, I/O interfaces and any other mechanism that can affect the PEs' timing behavior. From the perspective of how the shared memory is organised, there are three main categories, i.e. *centralised, distributed* and *mixed.* In *centralised* memory organisation, the time for any PE to access any memory location (typically via a bus) is uniform, when there is no interference, that is the time does not depend on the targeted memory location/address. On the other hand, in the *distributed* memory organisation, PEs use a different mechanism, such as *Direct Memory Access* (DMA) engines or NoC, to access various memory locations, thus having a *Non-Uniform Memory Access* (NUMA) in terms of timing. The *mixed* memory organisation is a combination of the *centralised* and *distributed* memory organisations, where a subset of PEs access their shared memory uniformly, but accessing the memory shared by another set of PEs is non-uniform.

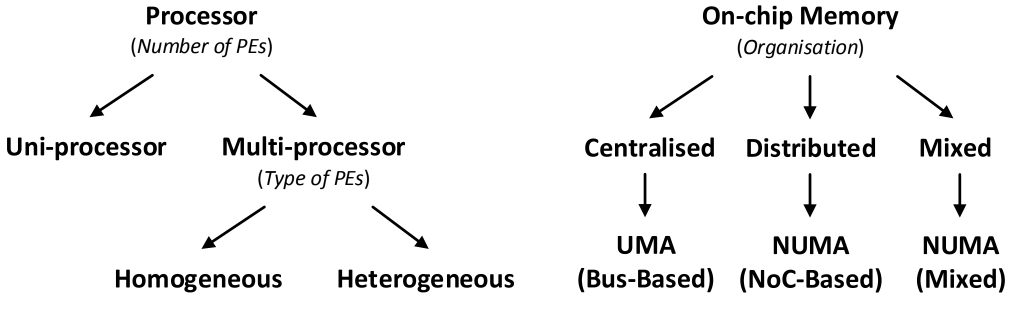


Figure 2.3 – Different types of processor architectures.

2.2.1 Homogeneous Architecture Types

The focus of this dissertation is on *homogeneous multi-processor* architectures, with the PEs being general purpose cores, for reasons explained in the Introduction. Following is a description of the homogeneous architectures covered in this dissertation according to their on-chip memory organisation. It must be pointed out that while the terms "centralised", "Uniform Memory Access", "distributed", "Non-Uniform Memory Access" frequently refer to main memory organisation, here are used for the on-chip memory.

Centralised architectures

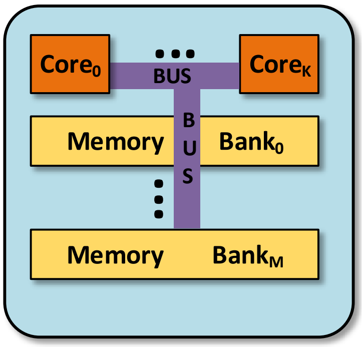


Figure 2.4 – A typical centralised multi-processor architecture.

In centralised homogeneous architectures, a number of general purpose PEs, henceforth cores, have a private cache memory (for instructions and data) but also share a part of the on-chip memory. The shared memory can be divided in several banks and cores can access a bank via a bus. Thus, in the general case, a bus connects a subset of the cores with a subset of the banks. To resolve conflicts of simultaneous requests for shared resources, i.e. buses, memory, these resources are arbitrated causing *timing interference*. Thus, a simultaneous memory request can be arbitrated at the bus arbiter, to resolve which core uses the bus first, and also at the memory arbiter, to resolve which memory request is served first by the memory controller of the memory bank.

For the purpose of interference analysis, it is assumed that a memory arbiter is dedicated to one memory bank. This is assumption is not restrictive as, in the case of multiple banks per memory arbiter, these banks can be considered as one. Also, it assumed that a continuous region on a memory bank can be allocated, as in sequential memory addressing, e.g. Kalray MPPA-256 [37], or strided memory allocation, e.g. CUDA [51].

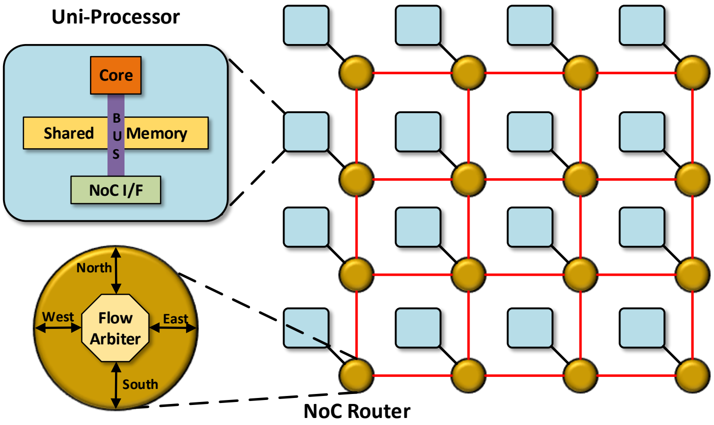


Figure 2.5 – A typical distributed multi-processor architecture.

Distributed architectures

In distributed homogeneous architectures several computing nodes are interconnected with a NoC. The computing nodes have a single core, with its private instruction and data caches, connected with a bus to the shared memory. In that sense, it is no different than a uni-processor. Additionally, a computing node has a NoC interface (NoC I/F) through which data are sent/received from other computing nodes. In order to parallelise data transfers, the NoC I/F can have multiple channels. A data transfer between a source and a destination computing node, the source NoC I/F (using its channels independently of what the core is executing) reads data from the shared memory, forms the network packets and injects them into the NoC, via a link to its corresponding NoC router. The packets are then forwarded from router to router, according to a routing policy, until they reach their destination. At the receiving end, the destination NoC I/F retrieves the data from the packets and places them into the destination shared memory, thus completing the data transfer.

Apart from the bus and memory arbiter that exist to resolve conflicts between the core and the NoC I/F of a computing node, there is an additional arbiter at each NoC router. For the purpose of interference analysis, it is assumed that when two packets simultaneously arrive at a NoC router from different input network links and are directed to the same output links, the flow arbiter decides which packet will be served first. As, the focus of this dissertation is in interference, it is also assumed that NoC routers will not block and cause back-pressure (which can be achieved by flow regulation at the source [36]). We consider that such assumption can be alleviated with proper adaptations of the interference analysis, but in principle should be handled by the deployment process and not the interference analysis.

Mixed architectures

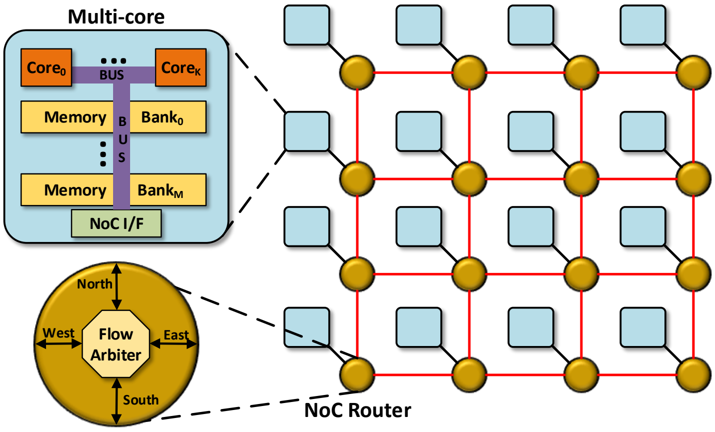


Figure 2.6 – A typical mixed multi-processor architecture.

Mixed homogeneous architectures are a combination of the centralised and distributed architectures. Their main difference from the distributed paradigm is that the computing node is replaced by a computing *cluster*, which essentially is a multi-processor with a NoC I/F. In terms of arbitration points, mixed architectures are the same as distributed architectures, i.e. they have memory, bus and flow arbiters. But, compared to distributed architectures, the number of sources of arbitrated requests, and by extension sources of timing interference, is larger.

Stemming out from the focus on homogeneous architectures, it is assumed that the computing clusters are identical in the sense that the time to execute a task does not depend on which computing cluster is executed.

2.2.2 Generic Architecture Model

In order to capture the different architectures with a single model that will be used in the proposed methods of this dissertation, the *generic architecture model* is introduced. A single model is important as it enhances applicability of the proposed methods, which in the context of hard real-time systems is desirable, as safety properties have to be proven only once.

Definition 2.2.1 (Generic architecture model)**.** A generic architecture model is a tuple $\mathcal{GA} = (\mathcal{C}, \mathcal{K}, \mathcal{L}, \mathcal{M}, \mathcal{N})$ where:

- $\mathcal{C}$ is a set of computing clusters
- $\mathcal{K}$ is a set of processing cores per cluster

- $\mathcal{L} \subseteq \mathcal{C} \times \mathcal{C}$ is a set of links among computing clusters that form the NoC
- $\mathcal{M}$ is a set of memory banks per cluster
- $\mathcal{N}$ is the set of NoC channels of a NoC I/F

When the generic architecture model is instantiated to a concrete architecture model matching with one of the homogeneous architecture types described earlier, i.e. (i) the centralised architectures $\mathcal{MA} = (\mathcal{C}, \mathcal{K}, \emptyset, \mathcal{M}, \emptyset)$ which do not have a NoC interconnect, (ii) the distributed architectures $\mathcal{DA} = (\mathcal{C}, \{k\}, \mathcal{L}, \mathcal{M}, \mathcal{N})$ with one core k per cluster, and (iii) the mixed architectures $\mathcal{MX} = (\mathcal{C}, \mathcal{K}, \mathcal{L}, \mathcal{M}, \mathcal{N})$.

For the architectures that have a NoC it is assumed that each cluster has one NoC I/F, with multiple channels, connected to a single dedicated NoC router [32] and has a separate bus to access the shared memory.

2.2.3 Kalray MPPA-256

The Kalray MPPA-256 chip consists of 256 processing VLIW cores grouped in 16 computing clusters with each cluster comprising 16 processing cores at $400MHz$ and $2MB$ of shared scratchpad memory (Figure 2.7). The shared memory consists of 16 independent $128kB$ memory banks with 64-bit words organised in two sides, *odd* and *even*. Each cluster has one NoC I/F, subdivided in transmit (Tx) and (Rx) interfaces, implemented with an a 8-channel DMA engine [38].

Each cluster NoC I/F is coupled with a dedicated NoC router, which are organised in a 2D-torus topology (Figure 2.8). Each router has 5 full duplex links, one for each direction plus one for its corresponding computing cluster. Each link to another router has an

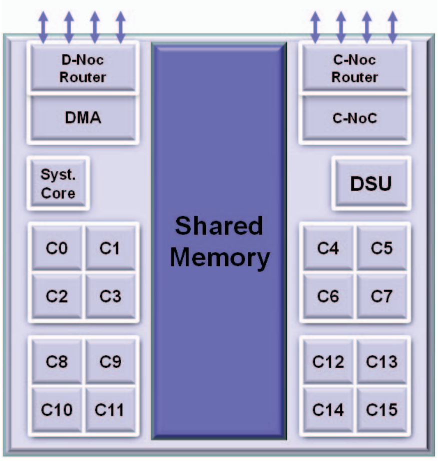


Figure 2.7 – Kalray MPPA-256 compute cluster [37].

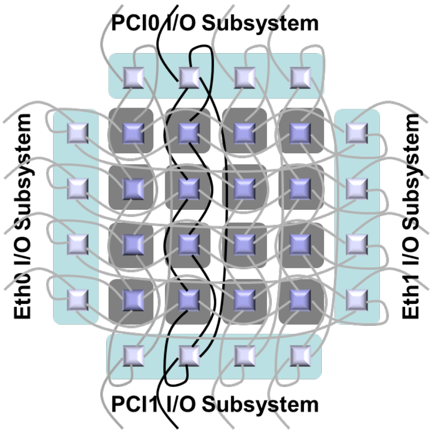


Figure 2.8 – Kalray MPPA-256 NoC topology [36].

independent round-robin arbiter and 4 FIFOs, one per input direction. The link to its dedicated cluster has one additional FIFO, so the cluster can send packets to itself. This means that the router can send back received packets from its cluster, but not from other routers.

Having this particular architecture that matches the generic architecture model $\mathcal{GA}$, the Kalray MPPA-256 can be reconfigured (depending on how an application is deployed) according to the different types of architectures considered. In specific the Kalray MPPA-256 can serve as:

- *Centralised architecture*: where only one computing cluster is utilised
- *Distributed architecture*: where only one core from each cluster is utilised
- *Mixed architecture*: where the full chip, i.e all 256 cores, are utilised

2.3 Application and System model

To provide real-time guarantees for an application, deployed on an architecture, a formal model for the application is also necessary. An application consists of a set of *computation tasks*. There is one significant distinction, encountered especially in the Scheduling Theory community, for the set of computation tasks. *Independent* tasks are those tasks which, if ready, can be scheduled in any order, while for *dependent* tasks there is an inherent scheduling order imposed. For example, compressing an image according to the JPEG standard requires that the image is split in blocks before it is transformed using the *Discrete Cosine Transformation* (DCT) method. So, from a task perspective, there is a data-dependency between the tasks that form the blocks and the DCT transformation tasks.

When all of these computation tasks are executed, the application is considered to have completed an *execution*, that is, the input data have been consumed and the desired output has been produced. The output is considered to be correct if during the execution there are no data-races, deadlocks etc. This dissertation considers applications that are composed of *data-dependent* computation tasks and are *iteratively executed* on a stream of input data, called *streaming* [109] or *data-flow* [68, 49] applications. This particular class of applications is suitable for architectures with large number of cores, as they exhibit a high degree of task- and data- parallelism.

An application composed of a set of *computation tasks* with dependencies can be modelled with various *Models of Computations* (MoCs), such as *Synchronous Data-Flow graphs* (SDFs) [68] and *Kahn Process Networks* (KPNs) [49]. Such models provide necessary correctness properties (such as deadlock freedom, absence of data-races, confluency, buffer protection, etc.) by-construction, if their semantics are respected. Without being strictly restricted to, an application will be modelled as a particular class of SDFs, called split-join graphs [106]. The choice of this particular model is due to (i) the explicit modelling of parallelisation factors and (ii) available tool-chain for these models.

Generally, an SDF model is a directed graph $SDF = (\mathcal{U}, \mathcal{E})$ with $\mathcal{U}$ being a set of computation *actors* and the binary relation $\mathcal{E}$ representing a set of FIFO channels among them.. In this model, *actors* communicate with each other by sending ordered streams of data elements (called tokens). When an actor is *fired* —which corresponds to the notion of task execution— it consumes a fixed amount of data tokens from its input FIFOs and produces a fixed amount of data tokens to its output FIFOs. These amounts are called production-/consumption *rates* and are defined for each of the actor's FIFOs. The production/consumption of data tokens are blocking operations, that is, if an input FIFO is empty (respectively an output FIFO is full) the corresponding read (respectively write) operation will be blocked until the FIFO contains enough data (respectively has enough free space).

Generally, it is considered acceptable in SDF models to fire an actor before input data and output space is available, as the actor would block when it would try to access a FIFO. Nevertheless, typical scheduling approaches generate schedules that avoid such situations. In our case where timing correctness is of importance, blocking of an actor would mean that its WCET would change. Thus, to guarantee correctness, a deployment solution must ensure that prior to firing an actor, there is enough data at its input FIFOs and enough free space at its output FIFOs. Otherwise, the actor would block, which would alter its WCET in isolation. Additionally, for efficient utilisation of the on-chip memory, it is assumed that actors can use their allocated space in their input/output FIFOs, to load/store intermediate results of their computation.

To guarantee that there is an *unbounded execution* of the SDF with *bounded FIFOs*, certain *consistency constraints* must be met. These are called balance equations [68] which essentially state that after a finite number of actor firings all data, which were produced, are consumed.

Definition 2.3.1 (SDF Consistency)**.** For each FIFO $e \in \mathcal{U}$, with $e = (u, u')$, let $\#_u, \#_{u'}$ be the number of times actors u, u' are fired, respectively. Also let r^e_{out} be the production rate of u and r^e_{in} the consumption rate u' for FIFO e; a SDF is *consistent* if there is a non-zero integer solution to the following *balance* equations, for all edges $e = (u, u') \in \mathcal{U}$:

$$\exists \#_u, \#_{u'} \in \mathbb{N}_+ : \quad \#_u * r^e_{out} = \#_{u'} * r^e_{in} \tag{2.3}$$

For consistent SDFs, the existence of *unbounded*, i.e. deadlock free, *execution* of that SDF with bounded FIFOs is decidable [67]. ■

There are several advantages of SDF models apart from guaranteeing correctness properties; SDF models can be used to model complex applications and allows to explore the latency-throughput-buffer trade-offs.

Given a solution to the balance equations, the *execution model* can be constructed, by deriving a task graph containing one task for each actor firing and a dependency relation that describes which firings of two dependent actors can be performed in parallel. This is essential to compute the set of tasks that can be executed in parallel with a given task, which affects the interference estimations in Chapter 3.

Definition 2.3.2 (Execution model)**.** An execution model $EM(E_S)$ of a directed acyclic task graph $G = (V, E)$ is its parameterised version $EM(E_S) = (V, E \cup E_S)$ where the data-dependency relation E is augmented with additional scheduling dependencies E_S, such that $EM(E_S)$ is also a DAG. ■

Definition 2.3.3. Given a consistent SDF graph $SDF = (\mathcal{U}, \mathcal{E})$ and the solution of the balance equations (the number of firings $\#_u$ for each actor $u \in \mathcal{U}$) and r^e_u (the rate of actor u for its FIFO e), the execution model $EM(\emptyset)$ with unbounded FIFOs is derived as follows:

- $V = \{v^i_u \mid \forall u \in \mathcal{U}, \forall i \in [1, \#_u]\}$, that is v^i_u denotes the i^{th} firing of actor u
- $E = \left\{(v^i_u, v^j_{u'}) \mid (u, u') \in \mathcal{E},\ \ (i-1) * r^{(u,u')}_u < j * r^{(u,u')}_{u'}\right\}$, that is, between two data-depend actors u, u', the task $v^j_{u'}$ cannot be executed before its input data from task v^i_u have been produced.

■

The dependencies of the execution model essentially prevent from FIFOs to underflow, by preventing a task v' to execute, if the task v that produces the input data of v' has not finished its execution. This is illustrated in the following example:

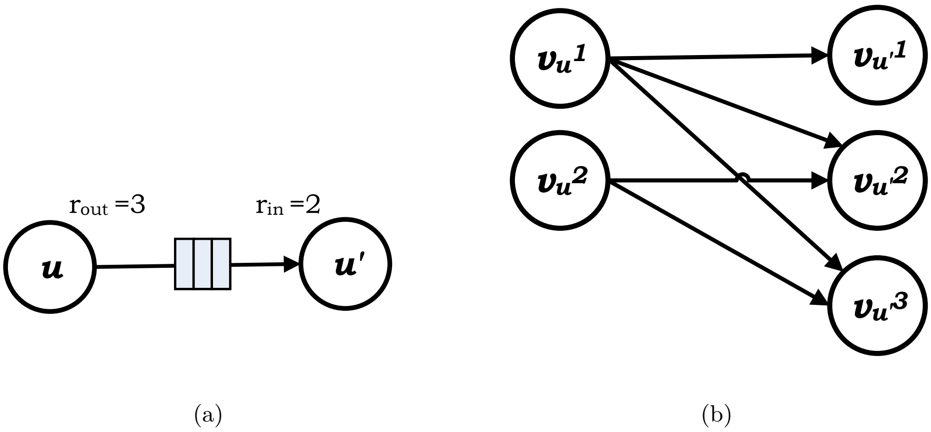


Figure 2.9 – Example of underflow protection through the dependency relation E of the execution model EM

Example 2.3.4. Consider the SDF graph of Figure 2.9a, with actor u producing three tokens at each firing and actor u' consuming two. In Figure 2.9b, the corresponding execution model is presented for the minimal solution of the balance equation of the SDF graph. The dependency relation constructed prevents all tasks of actor u', i.e. $v^1_{u'}, v^2_{u'}, v^3_{u'}$, to be executed before the first firing of actor u, i.e. v^1_u. The second firing of u' requires tokens from the first and the second firing of u, according to production/consumption rates, therefore task $v^2_{u'}$ is forced to execute after v^2_u. The same does not hold for the first firing of u', so task $v^1_{u'}$ can executed in parallel with the second firing of actor u. This is permitted in the execution model as task $v^1_{u'}$ does not depend on v^2_u. ■

2.3.1 Application Model

A split-join graph [106] $SJ = (\mathcal{U}, \mathcal{E}, \alpha)$ is a directed acyclic (DAG) SDF graph $(\mathcal{U}, \mathcal{E})$ annotated with a parallelisation function α, assigning a parallelisation factor $\alpha(e) \in \mathbb{Q}_+$ for each edge e of the graph. An edge e with parallelisation factor $\alpha(e) > 1$ is called *split*, with $\alpha(e) < 1$ is called *join* and with $\alpha(e) = 1$ is called *neutral*. Split-join graphs also require a *well-formedness* condition which states that "the product of parallelisation factors should be 1 for any path from a starting actor (i.e. with no predecessors) to an ending actor (i.e. with no successors) and the sequences of parallelisation factors of those paths respect the *matching-parenthesis* grammar[1]. This condition essentially ensures that all splits are joined in a meaningful manner, that is, all splits are eventually joined and that a split factor a is matched with a join factor $\frac{1}{a}$. For example consider the following sequences:

[1] The *matching-parenthesis*, a.k.a. matching-braces, is a classic problem in compilers.

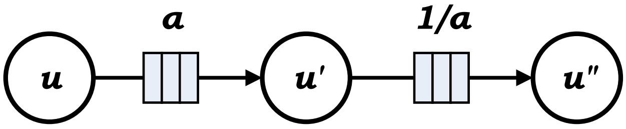


Figure 2.10 – A split join graph

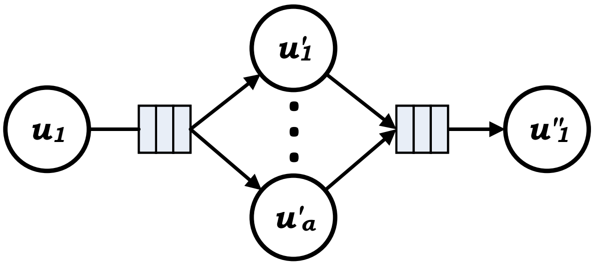


Figure 2.11 – Corresponding task graph, with arrows denoting the input/output FIFOs

- “$2 \cdot 3 \cdot 1/3 \cdot 1/2$” is well-formed
- “$2 \cdot 3 \cdot 1/2 \cdot 1/3$” is not balanced as 3 is followed by the the non-matching $1/2$
- “$2 \cdot 3 \cdot 1/3$” is not complete as the product of parallelisation factors is not 1

Split-join graphs are a natural variation of SDF graphs. In an SDF model a FIFO channel between two actors cannot be shared with another actor, but tasks originating from those two actors inherently share that FIFO. In a similar manner, in split-joint graphs functionally equivalent actors can access the same FIFO according to. A split-join graph $SJ = (\mathcal{U}, \mathcal{E}, \alpha)$ is *consistent* if there is a non-zero integer solution to the following *balance* equations, for all edges $e \in \mathcal{E}$:

$$\exists \#_u, \#_{u'} \in \mathbb{N}_+ : \quad \#_u * r^e_{out} = \#_{u'} * r^e_{in} * \alpha(e) \tag{2.4}$$

Intuitively, these balance equations relate the degree of parallelisation between the firing of the connected actors. For example consider two adjacent actors $e = (u, u')$, with u having a production rate $r^e_{out} = 4$, u' a consumption rate $r^e_{in} = 1$ and a parallelisation factor $\alpha(e) = 4$. This can be interpreted in multiple ways; tokens produced by a single firing of u will be consumed either by four firings of u', or two firings with double duration of that actor or by a single firing with quadruple duration.

Generally SDF and split-join models have no timing information or model the size of data exchanged between actors. Nevertheless, such information is important for hard real-time systems and WCET estimation. Thus, as in [105], we model an application as an annotated split-join graph, as follows:

Definition 2.3.5 (Application Model)**.** An application model is an annotated well-formed split-join graph $A = (\mathcal{U}, \mathcal{E}, \alpha, \delta_{iso}, \sigma)$ where:

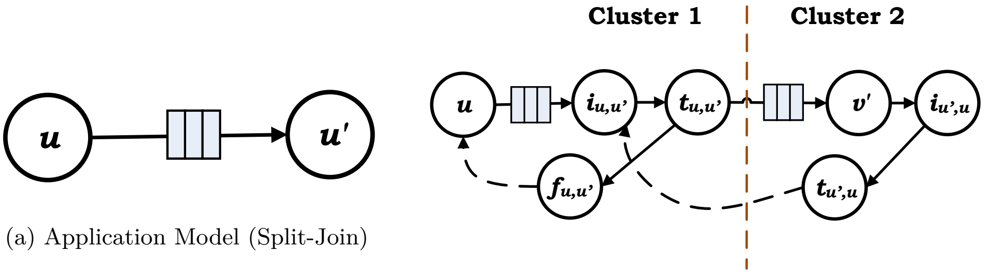
- $\mathcal{U}$ is the set of computation *actors*
- $\mathcal{E} \subseteq \mathcal{U} \times \mathcal{U}$ is the set of *FIFO channels*
- $\alpha : \mathcal{E} \to \mathbb{Q}_+$ is the parallelisation function
- $\delta_{iso} : \mathcal{U} \to \mathbb{N}_+$ is the WCET in isolation function which represents the *worst-case execution time* $\delta_{iso}(u)$ of single firing of an actor $u \in \mathcal{U}$ when executed in isolated environment with no interference
- $\sigma : \mathcal{E} \cup \mathcal{E}^{-1} \to \mathbb{N}_0$ is the *data-size* function, representing the memory requests each actor firing performs to FIFO e. That is, for a FIFO $e = (u, u')$ of two actors $u, u' \in \mathcal{U}$ then $\sigma\left(u, u'\right) = r^e_{out} * tk$ is the amount of memory requests that a single firing of u will perform when producing tokens of size tk Similarly, $\sigma\left(u', u\right) = r^e_{in} * tk$ is the amount of memory requests that a single firing of u' will perform when consuming tokens produced by u ■

2.3.2 System Model

While such a model is sufficient to describe the execution of an application on a multi-processor architecture, it does not fit architectures that have a NoC interconnect, as it does not capture the behavior of the NoC. A faithful model for the execution of an application on a generic architectures should account for the NoC transfers.

For this reason additional actors have to be introduced in the model. These are called *communication* actors, and model the three phases of a NoC communication. An *inter-cluster* data-exchange between actors u and u' consists of:

1. the NoC initialisation: where a core configures the NoC I/F with the length of the data-transfer along with the base memory addresses from where to fetch and where to deposit the data and the NoC channels that should be used for the transfer.
2. the NoC transfer: during which the NoC channel fetches the data from the memory, forms the packets and forwards them to the destination cluster over the NoC. The NoC I/F of the target cluster receives the data and places them in the destination memory.
3. the NoC finalisation: where the core that initialised the transfer polls the NoC interface to check if all the data have been transferred, so as to release the memory space occupied by the data.



(a) Application Model (Split-Join)

(b) System model (Split-Join)

Figure 2.12 – Example of a system model

The worst-case duration of the NoC initialisation and finalisation is assumed to be constant, while the duration of the NoC transfer depends on the amount of data and the distance between the source and destination clusters. It is assumed that NoC packets are not lost/dropped or that the WCET in isolation integrates the induced delays from such effects.

Such a system model can faithfully model the behavior of the system in terms of computation and communication. An application model A can be transformed into a *system model* if some partial knowledge is available about the mapping of actors. More precisely, to derive the system model, one would have to know in which cluster each actor is going to be executed. As an illustrative example of system model for a producer/consumer application model is illustrated in Figure 2.12. The producer and consumer are executed on different clusters, thus the have to utilise the NoC to exchange data. The dependency relation among the original actors and the, newly added, communication actors protect the FIFOs. Further details will be presented in Section 4 were the precise transformation is formally defined.

A consistent split-join graph $\mathcal{S} = (\mathcal{U}, \mathcal{E}, \alpha, \delta_{iso}, \sigma)$ that faithfully models the behavior of the system is called *system model*. Given a system model $\mathcal{S}$, the corresponding execution model can be derived, which will be used for our interference-sensitive WCET analysis.

2.3.3 Deployment

Having defined the fundamental models for an application, an architecture and an execution model, we know can formally define what constitutes a deployment and when a deploymen is considered safe.

Definition 2.3.6 (Deployment solution)**.** For a system model $\mathcal{S} = (\mathcal{U}, \mathcal{E}, \alpha, \delta_{iso}, \sigma)$ and its corresponding execution model $EM(\emptyset) = (V, E \cup \emptyset)$ a deployment to the architecture $\mathcal{GA} = (\mathcal{C}, \mathcal{K}, \mathcal{L}, \mathcal{M}, \mathcal{N})$ is the tuple $D = (\mu_K, \mu_M, \rho, \phi, \beta)$ where:

- the *task mapping* function $\mu_K : V \rightarrow \mathcal{C} \times \mathcal{K}$ assigns tasks to PEs
- the *memory mapping* function $\mu_M : \mathcal{E} \rightarrow \mathcal{C} \times \mathcal{M}$ maps FIFO channels to memory banks
- the *routing* function $\rho : T \rightarrow 2^{\mathcal{L}}$ which assigns to a transfer task $t \in T \subset V$ an acyclic path $(c_1, c_2), (c_2, c_3), \ldots, (c_{n-1}, c_n)$, of NoC links, upon which the transfer is routed
- the *begin* function $\beta : V \rightarrow \mathbb{N}_+$ which denotes the begin times of tasks ■

Definition 2.3.7 (Safe deployment)**.** Given a set of real-time constraints, i.e $u \in \mathcal{U},\ \epsilon(u) \leq d_u$, for an application $A = (\mathcal{U}, \mathcal{E}, \alpha, \delta_{iso}, \sigma)$ a deployment D is called *safe* iff all actors finish their firings before their deadlines, i.e. $\forall v_u^i \in V,\ \epsilon(v_u^i) \leq d_u$, and the FIFO channels remain protected. ■

In the following chapters we will show how a safe and efficient deployment solution can be derived. In particular, in the next Chapter we focus on how to improve the WCET estimations δ, based on the WCET in isolation δ_{iso}, by acquiring improved interference estimations, while on Chapter 4 we illustrate the proposed deployment process and in Chapter 5 the proposed runtime adaptation technique.

Χρόνου φείδου
— Χίλων ο Λακεδαιμόνιος

Do not waste time
— Chilon of Sparta

3 Interference-sensitive WCET

Acquiring the optimal *Deployment* (D) of a streaming application onto a generic architecture, providing hard real-time guarantees, requires to solve the deployment optimisation problem using the task Worst-Case Execution Time (WCET). Generally, the WCET of every task v is composed by its *WCET in isolation* (δ_{iso}), which includes delays to load/store data, and the delays due to *interferences* (ι) that tasks can experience when simultaneously access shared resources, e.g. buses/memories.

$$\delta_{\iota}(v) \quad = \quad \delta_{iso}(v) \; + \; \iota(v) \tag{3.1}$$

The accurate estimation of these interference delays depends on the deployment solution, which results in an inter-dependency between “WCET estimation” and “deployment optimisation”. Typically, this inter-dependency is avoided by over-approximating the WCET, such that the WCET estimations include all possible interferences. While this simplifies the problem at hand, it can lead to under-utilised systems or even render the deployment of an application to the target architecture infeasible.

The focus of this chapter is to present the benefits of interference analysis in acquiring more efficient deployments compared to traditional approaches. In order to acquire efficient deployments, it is imperative to break the inter-dependency between WCET estimation and deployment optimisation. Given an execution model, the proposed interference-sensitive method (*is*WCET) illustrated in Figure 3.1, breaks the inter-dependency by initially over-approximating the interference ι^{over} and deriving a safe deployment. The resulting WCET estimations $\delta_{\iota^{over}}$ are at most as pessimistic as the one derived with traditional methods.

Having a deployment solution, the method tightens the interference estimations ι^{over} by *spatio-temporal* exclusion, i.e. excluding interferences from tasks that do not overlap in time or share resources (space). Based on these newly estimated interferences (ι^{D}), tighter WCET estimations are computed ($\delta_{\iota^{D}}$), which will be referred as interference-sensitive

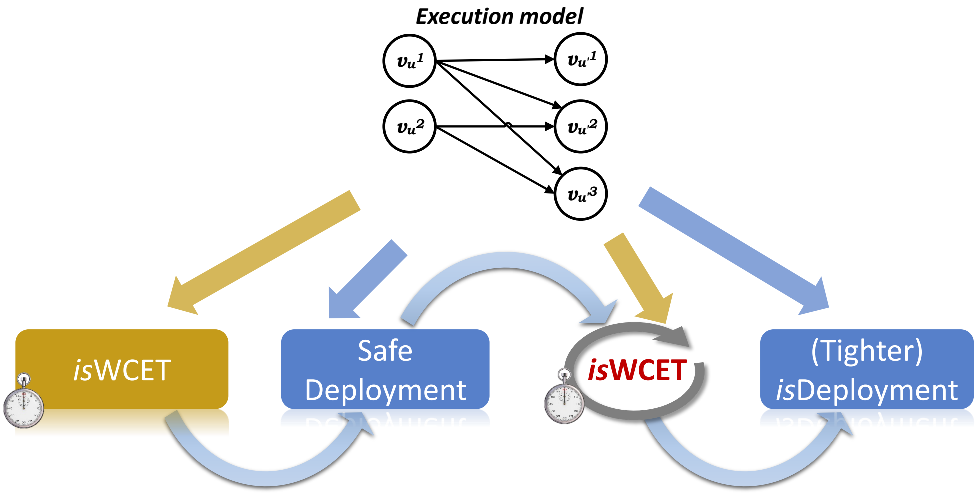


Figure 3.1 – Overview of the *is*WCET approach

WCET, or for short *is*WCET. These *is*WCET estimations are only valid for the particular deployment D, or for deployments D' such that the interference $\iota_{D'}(v)$ for each task v is not greater than $\iota_D(v)$, and still meet the given real-time constraints C_{RT}. In others words, while using interference-sensitive estimations enables realisation of efficient hard real-time systems, such systems are restricted in flexibility at runtime, in terms of rescheduling, memory management, etc. This is a known trade-off [76, 78], evidently also by the various types of resource allocation and real-time scheduling approaches (global, partitioned, with/without migrations, etc). We refer the reader for a thorough review [14, 82, 34, 25].

In the following sections of this chapter, we present the considered types of interference and arbitration policies, our *is*WCET analysis and the evaluation results for reference and real-life applications deployed on Kalray MPPA-256. The proposed *is*WCET method excludes on average 68%, yields an average improvement of the WCET upper bound by 5% to 10% for *all* tasks of all benchmarks and reduces overhead due to interference 44%, on average.

3.1 Interference Model

In a generic architecture $\mathcal{GA}$ there are three types of arbiters considered, that is (i) the *bus arbiter* (ii) the *memory controller* and (iii) the *NoC router*. Within a computing cluster, when two (or more) cores that share a bus simultaneously issue memory requests, these requests will be arbitrated by the bus arbiter based on a predetermined arbitration policy. Similarly, when two (or more) memory requests simultaneously arrive at a memory controller, the memory controller will arbitrate these request to resolve which will be served first. The same holds for NoC routers, that is when two (or more) packets arrive at the same time, or already exist in the NoC buffers, these packets will be arbitrated.

Recall that the there are two categories of tasks, *computation* (modelling the application) and *communication* (introduced to model task communication). Out of these tasks, only computation and transfer tasks utilise these arbitrated resources. Thus, the following types of interference can occur in a generic architecture:

- **Intra-cluster** interference, where the interfering tasks reside on the same computing cluster. This occurs when any number of tasks (any combination of computation and transfer tasks) are executed in parallel on the same computing cluster and share resources (memory banks, buses).
- **Inter-cluster** interference, where the interfering tasks are executed in parallel on different clusters. This category involves computation tasks of one cluster and transfer tasks of different clusters which are trying to access the same memory bank.
- **NoC** interference, where transfer tasks interfere with each other when their routes intersect at a NoC router.

In addition to the types of interference, the granularity of requests has to be modeled. Requests include, memory operations (reads, writes) that cores, or the NoC I/F, issue to the shared memory and network operations (transmit) that the NoC I/F issues to the shared NoC routers. We choose the granularity of the memory operations to be performed on a word-level and network operations to be performed on a packet-level. That is, for the purpose of interference analysis, data reads and writes to the shared memory are performed as a sequence of word-by-word memory operations. Similarly, network requests are performed as a sequence of packet-by-packet transfers. A single-word memory operation is called a *memory request* and a single-packet network operation is called a *network request*. This particular choice is based on what most processors support for a single memory/network request. Recall that the baseline delay for the bus to transmit the data, the memory controller to handle the request and the NoC to transmit the data, are all accounted in the WCET in isolation of the corresponding tasks. Thus, the aim is to estimate the additional time it would take for a task to complete its execution, in the worst-case, due to interference. As we assume no other information about the tasks, in the worst-case all requests will be single requests. Additional request burstiness information, if provided by the WCET in isolation method, can be leveraged to further improve these estimations.

It is considered that when a request interferes with other requests at any arbiter, the extra delays that will occur due to conflicts, are linearly proportional to the number of conflicting requests. That is, if $n+1$ requests conflict at the same time (and no other requests arrive until they are served), then in the worst-case no request will suffer a delay of more than $n * ad$, where ad is the single-conflict arbitration delay.

3.1.1 Hardware Arbitration Policies

Prior to presenting the interference analysis, the considered arbitration policies have to be presented. We consider the most commonly used arbitration policies that exhibit these interference delays. For example, we do not consider *Time-Division Multiple Access* (TDMA) arbiters, as the time for a request to be served depends on when the request was issued and not on the number of simultaneous requests. This means, that such resource partitioning techniques cannot benefit by an interference analysis. In particular, TDMA arbitration has the same behavior as a *Round-Robin* (RR) arbitration policy when it constantly receives requests from all the possible sources.

The interference delays introduced by the arbitration policies considered, i.e. round-robin and fixed priority, will be described on a task-centric manner, in the sense of "what is the worst-case delay task v can experience when executed in parallel with a set of tasks V". For each arbiter, the single-conflict arbitration delay $\boldsymbol{ad}$ – the time to serve another request – is assumed to be known and constant.

Definition 3.1.1 (Arbitration constants)**.** Given a set of arbitrated resources R, $\boldsymbol{ad}_r \in \mathbb{N}_+$ is the single-conflict arbitration constants (in cycles) for a resource $r \in R$. ■

In particular we consider that the arbitrated resources in the generic architecture $\mathcal{GA}$ are the buses, memory banks and NoC routers. That is, given the resource set $R = \{Bus, Mem, Rt\}$

- $\boldsymbol{ad}_{Bus} \in \mathbb{N}_+$ is the arbitration delay for a memory request to access a shared bus
- $\boldsymbol{ad}_{Mem} \in \mathbb{N}_+$ is the arbitration delay for a memory request to access a memory bank
- $\boldsymbol{ad}_{Rt} \in \mathbb{N}_+$ is the arbitration delay for a network request to traverse a NoC router

Round-Robin

In Round-Robin arbitration, the sources of requests are served in a cyclic fashion, each for a predetermined "quantum" of time.

Example 3.1.2. Consider three memory requests from two cores, that each would take at most that quantum of time to be served. In specific, core k_1 issues two memory requests, denoted as k_1^1 and k_1^2, and the arbitrated resource serves the first one, i.e. k_1^1. Before the shared resource completes serving the request, core k_2 issues also a memory request k_2^1. When request k_1^1 is served, the arbiter will choose to serve request k_2^1 from core k_2, despite the fact that request k_1^2 was issued earlier. Thus, the requests will be served in the following order:

$$k_1^1, k_2^1, k_1^2$$ ■

In this example, request k_1^1 does not suffer any delay, while requests k_2^1 and k_1^2 suffer once the arbitration delay ad, each. This is due to request k_2^1 waiting for request k_1^1 to complete and k_1^2 waiting for k_2^1. Notice that while request k_1^2 was served third, it only experiences a delay of ad, since the first request was from the same core. We assume that a core cannot interfere with itself, therefore that additional delay should be accounted in the WCET in isolation.

Any request that would to take more than the predefined quantum of time to be served, i.e. ad, is regarded as multiple simultaneously issued requests, with each request having a duration equal to the quantum of time. Thus, let $\sigma_r(v)$ be the number of requests of task v to resource r, which is arbitrated in a round-robin fashion. Since round-robin is a fair arbitration policy there is a theoretical maximum of arbitration delays, any task can experience. For example, if resource r is shared by a set of cores K, then for any task v the maximum delay is $ad * (|K| - 1) * \sigma_r(v)$. That is, every request from task v will wait for $|K| - 1$ requests from the rest of the cores before that request is served.

Considering the previous example, though, the requests from both cores will suffer at most one arbitration delay ad, in all possible execution scenarios. Yet, the theoretical maximum delay for core k_2 is $2 * ad$, as it issues two requests and shares the resource with one core. This can lead to an non-neglible over-approximation for a large number of requests.

To avoid such an over-approximation, the delay due to interference can be defined on a per-task basis. That is, if task v is potentially executed in parallel with a set of tasks V which use resource r, the maximum amount of delay due to interference from that resource r is:

$$\iota_{ad_r}^{RR}(v, V) = ad_r * \sum_{v' \in V} \min\left(\sigma_r(v), \sigma_r(v')\right)$$

The rationale for this formula is that when two tasks with unequal number of requests are arbitrated by a round-robin arbiter, only the smallest amount of requests of the two tasks needs to be arbitrated.

Since the set of potentially parallel tasks V can grow arbitrarily large, the delay due to interference is defined as the minimum of the per-task delay and the theoretical maximum.

Definition 3.1.3 (Round-Robin interference delay)**.** For a task v that can potentially interfere with a set of tasks V on a resource r arbitrated with round-robin policy the worst-case interference delay is:

$$\iota_{ad_r}^{RR}(v, V) = ad_r * \min\left((|K| - 1) * \sigma_r(v), \sum_{v' \in V} \min\left(\sigma_r(v), \sigma_r(v')\right)\right) \tag{3.2}$$

Essentially, the intereference delays exprerienced by a task v, due to Round-Robin arbitration, is linearly proportional to the number of parallel request up to the theoretical maximum, where the delays do not increase regardless of the ammount of requests.

Fixed-Priority

In *Fixed-Priority* (FP) arbitration of a resource r, some requests are always served before others depending on their source. Following Example 3.1.2, if requests from core k_1 are given priority over requests from core k_2, then the requests will be served in the following order:

$$k_1^1, k_1^2, k_2^1$$

whereas if requests from k_2 are given priority over requests from core k_1 and all the requests arrive simultaneously, the requests will always be served in the following order:

$$k_2^1, k_1^1, k_1^2$$

In both these cases, the requests with higher priority experience no delays, as they are served immediately. Notice that in the case of fixed-priority arbitration there is no theoretical bound for the delay of low-priority requests, as requests of higher priority can be always issued before a lower priority request is served, thus leading to starvation.

Definition 3.1.4 (Fixed-priority interference delay)**.** For a task v that can potentially interfere with a set of higher priority tasks V on a resource r arbitrated with fixed-priority policy, the worst-case interference delay is:

$$\iota_{ad_r}^{FP}(v, V) = ad_r * \sum_{v' \in V} \sigma_r\left(v'\right) \tag{3.3}$$

3.2 Interference Analysis

Having defined the interference model and arbitration policies and given an execution model $EM(E_S) = (V, E \cup E_S)$, the interference for its tasks can be computed. This is achieved by computing *interference sets* for the tasks of the execution model $EM(E_S)$. An interference set of a task v for a resource r is the set of tasks that can potentially be executed in parallel with v and share the resource r. The set of potentially parallel tasks are computed based on the dependency relation of the execution model $EM(E_S$ and are improved when the deployment D is acquired, based on the begin/end functions β,ϵ)

Thus, in order to acquire tight interference estimations the *is*WCET method performs the following steps:

1. Generate *interference sets* based on $EM(\emptyset)$, assuming potentially parallel tasks interfere at all of their common resources.
2. For every task v, derive the interference delays $\iota^{over}(v)$ (up to the theoretical maximum) based on these over-approximated interference sets.
3. For every task, derive over-approximated WCET $\delta_{\iota^{over}}(v)$ (according to Equation 3.1).
4. Given a safe deployment solution $D = (\mu_K, \mu_M, \rho, \phi, \beta)$ iteratively reduce the *interference sets* by applying *spatio-temporal* exclusion according to the deployment solution D.
5. Derive tighter interference delays $\iota^D(v)$ and the interference-sensitive $\delta_{\iota^D}(v)$ (*is*WCET) estimations.

At the middle of the method, the deployment solution is required which is provided by the method in Chapter 4 or by any conventional methods. At the end of the process, the *is*WCET estimations are computed, which can be used to provide a more efficient deployment solution $D' = (\mu_K, \mu_M, \rho, \phi, \beta')$ which is also the focus of the next chapter.

The aim of this chapter, is to find the interference delays for the tasks of execution model EM of a system model S for application A. Recall that only the computation and transfer tasks can experience and cause interference. Let the set of these computation task be denoted as $V_A \subseteq V$ (application tasks) and the set of transfer tasks denoted as $T \subset V$. For any computation task v, let M_v, B_v and T_v be *interference sets* of tasks potentially conflicting with v. Specifically:

- $M_v \subset V_A \cup T$ is set of tasks (from the same cluster as v, hence *intra-cluster*) that can potentially interfere on a memory bank with v.
- $B_v \subset V_A$ is set of computation tasks (from the same cluster as v, hence *intra-cluster*) that can potentially interfere on a bus with v.
- $T_v \subset T$ is set of transfer tasks (from other clusters, hence *inter-cluster*) that can potentially interfere on a memory with v.

Using these interference sets, the interference delays a computation task v can experience is:

$$\iota(v) = \iota_{Mem}^{AP_{Mem}}(v, M_v) + \iota_{Bus}^{AP_{Bus}}(v, B_v) + \iota_{Comm}^{AP'_{Mem}}(v, T_v) \tag{3.4}$$

where $\iota_{Mem}^{AP_{Mem}}, \iota_{Bus}^{AP_{Bus}}, \iota_{Comm}^{AP'_{Mem}}$ are functions that are defined (in following subsections) for intra-cluster (memory/bus) requests and inter-cluster (memory) requests, respectively, based on the arbitration policies $AP_{Mem}, AP_{Bus}, AP'_{Mem} \in \{RR, FP\}$.

Similarly, the interference delay for a transfer task t is:

$$\iota(t) \;=\; \iota_{Mem}^{AP_{Mem}}(t, M_t) \;+\; \iota_{Rt}^{AP_{Rt}}(t, R_t) \;+\; \iota_{Comm}^{AP'_{Mem}}(t, T_t) \tag{3.5}$$

where $\iota_{Mem}^{AP_{Mem}}, \iota_{Rt}^{AP_{Rt}}, \iota_{Comm}^{AP'_{Mem}}$ are the respective functions for the interference t can experience when reading from the memory of the source cluster, traversing NoC routers and writing at the memory of the target memory.

In the following subsections, the exact definitions of interference sets and interference functions are presented along with their instantiation for Kalray MPPA-256. Prior to that, some essential formalism is presented regarding potentially parallel tasks, sharing of resources and temporal overlapping of tasks.

Definition 3.2.1 (Spatio-temporal predicates)**.** Given the begin function β of a deployment D for an execution model $EM(V, E)$, for any two tasks v, v' the $overlap(v, v')$ predicate evaluates to true, iff the execution of the tasks overlaps in time, i.e.:

$$overlap(v, v') = \max\big(\beta(v), \beta(v')\big) \leq \min\Big(\beta(v) + \delta_\iota(v), \beta(v') + \delta_\iota(v')\Big)$$

Similarly, for a resource r the $share_r(v, v')$ predicate evaluates to true, iff the tasks share the resource r. If the deployment D is not known, they automatically evaluate to true, only if it is possible for the tasks to be executed in parallel or share resource r, respectively. ■

Since the exact definition of the $share_r$ depends on the architecture, it will be instantiated in the following subsections for each resource $r \in \{Mem, Bus, Rt\}$

For any task v, the set of tasks that can interfere with that task, are those tasks that are executed in parallel and share a resource. Whether the scheduling of tasks is known or not, directly affects that set of parallel tasks. Tasks that transitively depend on v, or v depends upon, cannot possibly be executed in parallel. The rest of the tasks can be *potentially* be executed in parallel (if the scheduling is not known).

Definition 3.2.2 (Parallel tasks)**.** Given an execution model $EM(E_S) = (V, E)$, let P_v be the set of tasks that can be executed in parallel with task v is:

$$P_v = \{v' \in V \mid (v, v'), (v', v) \notin E^* \wedge overlap(v, v')\} \tag{3.6}$$

where E^* is the transitive closure of the dependency relation E. Also its projection P_{v_c} is defined according to Table 3.1 and its complement $\overline{P_{v_c}} = P_v \setminus P_{v_c}$

Notice that, if the deployment solution D is not known, the set of parallel tasks is over-approximated, as the $overlap$ predicate automatically succeeds, but still data-dependant tasks are always excluded. When the deployment solution D is provided, this set is

Table 3.1 – Projections of a task-set V, a dependency relation E and an execution model $EM = (V, E)$

Per cluster $c \in \mathcal{C}$	$V_c \overset{def}{=} \{v \in V \mid \pi(v) = c\}$	$E_c \overset{def}{=} E \cap (V_c)^2$	$EM_c \overset{def}{=} (V_c, E_c)$
Per core $k \in K$	$V_k \overset{def}{=} \{v \in V \mid \mu_K(v) = k\}$	$E_k \overset{def}{=} E \cap (V_k)^2$	$EM_k \overset{def}{=} (V_k, E_k)$

iteratively reduced until a fix-point is reached. The reason for the iterative reduction is due to the fact that the $\mathtt{overlap}$ *predicate depends on the interference estimations* ι. *Thus less overlaps leads (potentially) to less interference, which in turn can lead to even less overlaps, etc. The same holds for the interference sets* M_v, B_v, T_v, R_v *of any task* v, *since they will be computed based on the set of parallel tasks* P_v.

3.2.1 Intra-cluster Interference

Interference that originates within a particular cluster occurs on memory banks and bus arbiters from computation tasks and transfer tasks that transmit data to another cluster. The precise definition of the interference functions $\iota_{Mem}^{AP_{Mem}}(v, M_v), \iota_{Bus}^{AP_{Bus}}(v, B_v)$ and their interference sets are presented, along with examples.

To begin with, let us consider the interference delays that any task v will experience on the memory banks of its cluster. These are defined by function $\iota_{Mem}^{AP_{Mem}}(v, M_v)$ where essentially task v accesses a set of memory banks $\{m\}$. From the set of tasks P_{v_c} that can be executed in parallel with v, only the set of tasks M_v that share at least on memory bank can interfere. This is formally expressed as:

$$\iota_{Mem}^{AP_{Mem}}(v, M_v) = \sum_{m \in \mathcal{M}} \iota_{ad_m}^{AP_{Mem}}(v, M_v) \quad \text{with} \quad M_v = \{v' \in P_{v_c} \mid share_{Mem}(v, v')\} \tag{3.7}$$

where $ad_m = ad_{Mem}$ for all memory banks m as we are considering homogeneous architectures. By evaluating the interference function $\iota_{ad_m}^{AP_{Mem}}(v, M_v)$ according to the arbitration policy AP_{Mem}, the estimated amount of interference delays is acquired (see Example). Notice that $\iota_{ad_m}^{AP_{Mem}}(v, M_v)$ always evaluates to zero if task v does not access memory m.

Example 3.2.3. Consider two tasks v_1, v_2 that are potentially parallel with each other, and with no other. Also, let v_1, v_2 access the memory banks m_1, m_2 and m_2, m_3, respectively, which are arbitrated with a round-robin policy. The interference delays that v_1 will suffer from memory banks, using Equation 3.2 for the round-robin policy, are:

$$\iota_{Mem}^{RR}(v_1, \{v_2\}) = \sum_{m \in \mathcal{M}} \iota_{ad_m}^{RR}(v_1, \{v_2\}) = \sum_{m \in \mathcal{M}} ad_m * \sum_{v' \in \{v_2\}} \min\bigl(\sigma_m(v_1), \sigma_m(v_2)\bigr)$$

Since tasks v_1, v_2 only share memory bank m_2 and $\sigma_m(v)$ evaluates to zero if task v does not access memory bank m, the equation reduces to the expected:

$$\iota^{RR}_{Mem}(v_1, \{v_2\}) = ad_{m_2} * \min\left(\sigma_{m_2}(v_1), \sigma_{m_2}(v')\right) \qquad \blacksquare$$

In a similar manner, the interference delays that a computation task v can experience at the bus arbiter when tries to access its memory banks are:

$$\iota^{AP_{Bus}}_{Bus}(v, B_v) = \sum_{m \in \mathcal{M}} \iota^{AP_{Bus}}_{ad_{Bus}}(v, B_v) \quad \text{with} \quad B_v = \left\{v' \in P_{v_c} \cap V_A \mid share_{Bus}\left(v, v'\right)\right\} \tag{3.8}$$

Kalray MPPA-256

In Kalray MPPA-256, processing cores are organised in pairs, each pair shares two data-buses, one for each of the memory sides (*odd* and *even*). In this configuration, when one core of a pair accesses one memory side and the other core of the same pair accesses the other memory side, there is no conflict [37].

Thus, the $share$ predicates are defined as follows:

$$share_{Bus}\left(v, v'\right) = \begin{cases} \mu_K(v) = \lfloor \frac{\mu_K(v')}{2} \rfloor \wedge m\%2 = m'\%2 & \forall m \in \mu_M(v), \forall m' \in \mu_M(v') \\ true & \text{if } \mu_K \text{ or } \mu_M \text{ is unknown} \end{cases}$$

$$share_{Mem}\left(v, v'\right) = \begin{cases} \mu_M(v) \cap \mu_M(v') \neq \emptyset & \\ true & \text{if } \mu_M \text{ is unknown} \end{cases}$$

The bus arbiter implements a round-robin policy among requests coming from each instruction and data cache. The memory bank arbiter also follows a round-robin policy among all intra-cluster requests, e.g. from processing cores and NoC transmit (NoC Tx).

3.2.2 Inter-cluster Interference

As opposed to intra-cluster, inter-cluster interference is caused by tasks that are located in other clusters. That is for a cluster c, transfer tasks that are executed on a different cluster but have as destination that cluster c. Notice that such tasks can interfere with both computation and transfer tasks in cluster c. Thus, the set of parallelly executed transfer tasks with a task v, located in a cluster other than c, is:

$$T_v = \left\{t \in \overline{P_{v_c}} \cap T \mid share_{Mem}(v, t)\right\} \tag{3.9}$$

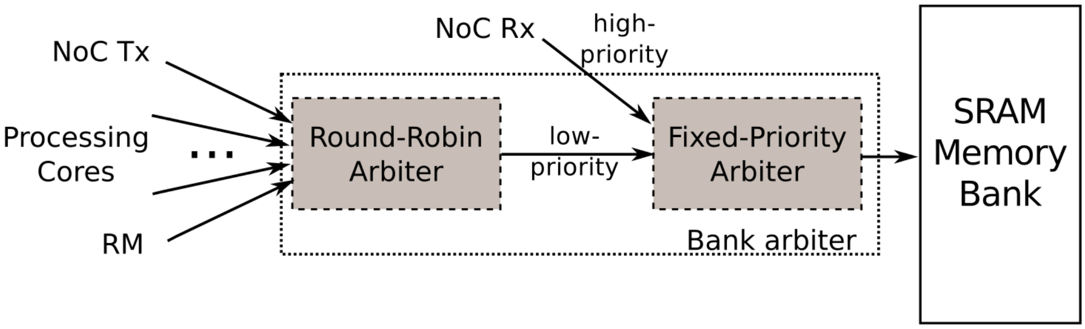


Figure 3.2 – Kalray MPPA-256 memory bank arbitration [48]

Kalray MPPA-256

In Kalray MPPA-256, memory accesses from incoming NoC flows have priority over other memory requests (Figure 3.2). That is, the NoC Rx interface is always given priority over the the intra-cluster memory request. (The $share_{Mem}$ for Kalray MPPA-256, required to compute the set of parallel transfer tasks was defined in the previous subsection)

Example 3.2.4. Consider two tasks, a computation task v_A and a transfer t that writes to cluster $\mu_K(v_A)$ such that those tasks are potentially parallel with each other, and with no other. Also, let v_A, t access the memory banks m_1, m_2 and m_2, m_3, respectively, of cluster $\mu_K(v_A)$ which are arbitrated with a fixed-priority policy (incoming transfers have priority). The interference delays that v_A will suffer from memory banks, using Equation 3.3 for the fixed-priority policy, are:

$$\iota^{FP}_{Mem}(v_A, \{t\}) = \sum_{m \in \mathcal{M}} \iota^{FP}_{ad_m}(v_A, \{t\}) = \sum_{m \in \mathcal{M}} ad_m * \sum_{v' \in \{t\}} \sigma_m(v')$$

Since tasks v_A, t only share memory bank m_2 and $\sigma_m(t)$ evaluates to zero if task t does not access memory bank m, the equation reduces to the expected:

$$\iota^{FP}_{Mem}(v_A, \{t\}) = ad_{m_2} * \sigma_{m_2}(t)$$

3.2.3 NoC Interference

Arbitration on NoC routers in the assumed model resembles that of Kalray MPPA-256, illustrated in Figure 3.3. In the assumed model each link is arbitrated independently, thus flow interference with each other only when two flows merge in one i.e. coming from different directions but continuing on the same direction. For ease of notation, it is assumed that flows can merge once (e.g. when routed with XY-routing). Recall, that it is also assumed that NoC routers will not block and cause back-pressure.

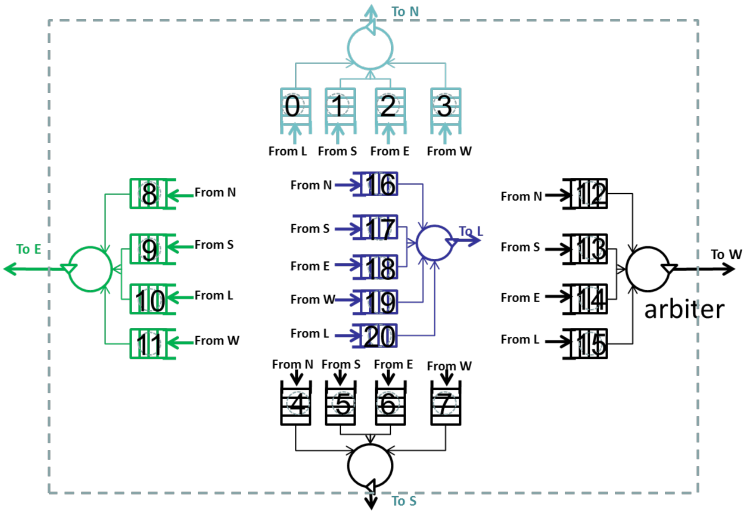


Figure 3.3 – Kalray MPPA-256 NoC router arbitration [36]

Thus, the interference a transfer task t can experience is:

$$\iota_{Rt}^{AP_{Rt}}(t,R_t) = \iota_{ad_{rt}}^{AP_{Rt}}(t,R_t) \quad \text{with} \quad R_t = \left\{t' \in P_t \mid share_{Rt}\left(t,t'\right)\right\} \tag{3.10}$$

$$share_{Rt}\left(t,t'\right) = \begin{cases} \rho(t) \cap \rho(t') \neq \emptyset & \\ true & \text{if } \rho \text{ is unknown} \end{cases}$$

Kalray MPPA-256

In Kalray MPPA-256, the NoC links form a 2D-torus as was depicted in Figure 2.8. In a NoC router there are five arbiters, one for each outgoing direction a flow can take, with each having a separate FIFO buffer for each incoming direction. The arbiters enforce a round-round policy seperately for each possible direction link (North, South, West, East). Since each link is seperately arbitrated, a flow will suffer interference delays from another flow only if they share a link on a NoC router.

Example 3.2.5. Consider three transfer tasks t_1, t_2, t_3 that are potentially parallel with each other, and the flows of transfer tasks t_2 and t_3 merge with the flow of t_1 (at different parts of the flow of t_1) and not among them. The interference delays that t_1 will suffer at the NoC, using Equation 3.2 for the round-robin policy, are:

$$\iota_{Rt}^{RR}(t_1,\{t_2,t_3\}) = ad_{Rt} * \sum_{t' \in \{t_2,t_3\}} \min\left(\sigma_{Rt}(t_1), \sigma_{Rt}\left(t'\right)\right)$$

Similarly the the interference delays that t_2 will suffer:

$$\iota_{Rt}^{RR}(t_2, \{t_1\}) = ad_{Rt} * \sum_{t' \in \{t_1\}} \min\left(\sigma_{Rt}(t_2), \sigma_{Rt}\left(t'\right)\right)$$

3.2.4 Safety

As the proposed method is aimed at hard real-time systems, it is of utmost importance to provide formal proofs regarding the safety of the method. A safe WCET estimation $\delta(v)$ for any task v is safe iff it is not less that the actual WCET, i.e. $\forall v : \delta_{act}(v) \leq \delta(v)$. Notice that, while the actual WCET $\delta_{act}(v)$ might not be known, it is still possible to prove that the *is*WCET $\delta_{\iota_D}(v)$ estimation is an upper bound for the actual WCET $\delta_{act}(v)$.

To prove the safety of the proposed method, it is necessary first to establish that the interference analysis does not exclude any possibly interfering tasks, a result upon which the proof will be based.

Lemma 3.2.6. For a given task v of an execution model $EM(E_S) = (V, E)$, each valuation of the interference sets M_v, B_v, T_v, R_v excludes only non-interfering tasks.

Proof. Two tasks v, v' can interfere iff their executions overlap and they share a resource. The interference sets select all the tasks from the set of parallel tasks P_v with which task v can possibly share a resource (according to Equations 3.7, 3.8, 3.9, 3.10). By definition of the $overlap$ predicate only non-overlapping, and thus non-interfering tasks, are excluded from the set of parallel tasks P_v. According to definitions of the $share_r$ predicates only tasks that do not share a resource with v are excluded from the set of parallel tasks P_v. Thus each valuation of the interference sets excludes only non-interfering tasks. □

Theorem 3.2.7 (Safety)**.** *The* *is*WCET *method is safe, i.e. for any task v of a given execution model $EM(E_S) = (V, E)$, the* is*WCET estimation $\delta_\iota(v)$ is at least as large as the actual WCET $\delta_{act}(v)$.*

Proof. Since the WCET time of any task is composed by its WCET in isolation ($\delta_{iso}(v)$) and its interference ($\iota(v)$) it is sufficient to prove that any interference estimation $\iota(v)$ is at least as large as the actual interference $\iota_{act}(v)$. It is also known, by Lemma 3.2.6, that interference sets do not exclude any interfering task and the considered arbitration policies make the worst case assumption. As an immediate consequence, interference estimation $\iota(v)$ cannot be less than the the actual interference $\iota_{act}(v)$, for any task $v \in V$. □

Recall, also, that when a deployment D for an execution model $EM(E_S) = (V, E)$ is provided to the *is*WCET method, the method iteratively reduces the interference estimations. It is also important to prove that the method eventually converges. The proof of convergence is founded upon the fact that the estimated interfence for each resource cannot increase if the corresponding set is reduced.

Lemma 3.2.8. For any task υ, the interference functions $\iota_{Mem}^{AP_{Mem}}(\upsilon, V)$, $\iota_{Bus}^{AP_{Bus}}(\upsilon, V)$, $\iota_{Comm}^{AP'_{Mem}}(\upsilon, V)$, $\iota_{Rt}^{AP_{Rt}}(\upsilon, V)$ are monotonic (non-decreasing) with respect to V.

Proof. Both arbitration policies $\iota_{ad_r}^{RR}(\upsilon, V), \iota_{ad_r}^{FP}(\upsilon, V)$ according to Equations 3.2, 3.3, are non-decreasing with respect to V. As a sum of non-decreasing functions, the interference functions $\iota_{Mem}^{AP_{Mem}}(\upsilon, V)$, $\iota_{Bus}^{AP_{Bus}}(\upsilon, V)$, $\iota_{Comm}^{AP'_{Mem}}(\upsilon, V)$, $\iota_{Rt}^{AP_{Rt}}(\upsilon, V)$ are also non-decreasing. □

Essentially, the method starting from an initial interference estimation, when $D = (\mu_K, \mu_M, \rho, \beta, \epsilon)$ is provided, iteratively improves the estimations by reducing the interference sets $M_\upsilon, B_\upsilon, T_\upsilon, R_\upsilon$. That is, the following sequence of computations are performed for each task υ starting from iteration 0: $\iota^0(\upsilon) \to \delta_{\iota^0}(\upsilon) \to \epsilon^0(\upsilon) \to (M_\upsilon^0, B_\upsilon^0, T_\upsilon^0, R_\upsilon^0) \to \iota^1(\upsilon) \to \ldots \to \iota^i(\upsilon) \to \ldots \to \iota^D(\upsilon)$

Theorem 3.2.9 (Interference improvement)**.** *For any task $\upsilon \in V$ at iteration i, the interference estimation $\iota^i(\upsilon)$ and the WCET estimation $\delta_{\iota^i}(\upsilon)$ of task υ are lower than or equal to the interference $\iota^{i-1}(\upsilon)$ and WCET estimation $\delta_{\iota^{i-1}}(\upsilon)$, respectively, of the previous iteration.*

Proof. Let us assume that there exists a task $\upsilon \in \upsilon$ such that $\iota^i(\upsilon) > \iota^{i-1}(\upsilon)$; then by applying Equations 3.4,3.5 we deduce that at least one of the following holds:

$$\iota_{Mem}^{AP_{Mem}}(\upsilon, M_\upsilon^i) > \iota_{Mem}^{AP_{Mem}}(\upsilon, M_\upsilon^{i-1})$$
$$\iota_{Bus}^{AP_{Bus}}(\upsilon, B_\upsilon^i) > \iota_{Bus}^{AP_{Bus}}(\upsilon, B_\upsilon^{i-1})$$
$$\iota_{Comm}^{AP'_{Mem}}(\upsilon, T_\upsilon^i) > \iota_{Comm}^{AP'_{Mem}}(\upsilon, T_\upsilon^{i-1})$$
$$\iota_{Rt}^{AP_{Rt}}(\upsilon, R_\upsilon^i) > \iota_{Rt}^{AP_{Rt}}(\upsilon, R_\upsilon^{i-1})$$

By monotonicity of the interference functions we conclude that either $M_\upsilon^i \supset M_\upsilon^{i-1}$, or $B_\upsilon^i \supset B_\upsilon^{i-1}$, or $T_\upsilon^i \supset T_\upsilon^{i-1}$, or $R_\upsilon^i \supset R_\upsilon^{i-1}$ holds, which contradicts with Lemma 3.2.6. Since the interference estimations cannot increase, the WCET estimations cannot increase either (according to Equation 3.1) □

Theorem 3.2.10 (Convergence)**.** *The iterative method will eventually converge.*

Proof. Essentially the method computes a fix-point of the interference sets, and by extension of the interference estimation. Since interference sets do not increase in size, the method will converge. □

Table 3.2 – Kalray MPPA-256 arbitration constants

Memory arbiter ad_{Mem}	7 cycles
Bus arbiter ad_{Bus}	4 cycles
NoC router ad_{Rt}	22 cycles
Initialization delay id_{NoC}	1000 cycles
Polling delay pd_{NoC}	100 cycles

3.3 Evaluation

3.3.1 Experimental Setup

In order to evaluate the applicability and benefits of the isWCET analysis method, we conducted experiments on a subset of StreamIt [109] benchmarks and a JpegDecoder (same benchmark used in [104]for comparison. The benchmark set used consists of 8 distinct applications, differing in the amount of computation for a single element of input, memory requirements and access patterns. Specifically, the sorting algorithms (InsertionSort, MergeSort and CompCount) have complicated memory access patterns, but are not as computationally heavy, compared to Beamformer, DCT and JpegDecoder. We also modelled DCT in 10 different ways, to study the impact of different levels of model parallelism. The size of the set of tasks ranges, for the aforementioned benchmarks, from 6 tasks up to 200.

As there is no publicly available tool (for Kalray MPPA-256) that can provide the WCET in isolation for the tasks of the applications of the benchmark, the WCET in isolation was obtained by profiling the tasks on a single core of the Kalray MPPA-256 (SDK version 1.4.1), with disabled caches and prefetch buffers. The amount of memory requests is statically predetermined according to the FIFO token size required and the number of token accesses each task performs. Additionally, in order for the profiling and the results to be as accurate as possible, all FIFOs and their tokens are aligned to memory boundaries, i.e. in memory address in multiples of a word, which affects the number of memory requests. This alignment guarantees that requesting any single-word-sized data request will not result into two single-word-sized memory requests.

The arbitration constants used for the Kalray platform are summarised in Table 3.2 and are based on publicly available information [35, 37, 104, 48] or experimental evaluation.

In order to study the benefits of using isWCET estimations, the benchmarks are first analysed without the knowledge of the deployment solution D, acquiring the over-approximated WCET δ^{over}. Subsequently, the benchmarks are analysed by providing the deployment solution D, which is constructed using the method proposed in Chapter 4.

Table 3.3 – Worst-case latency (in **Kcycles**) of StreamIt benchmarks deployed on Kalray MPPA-256 with cores operating at 400 MHz, based on the over-approximated WCET estimations δ^{over}

Bench.	Tasks	Lat.	Bench.	Tasks	Lat.	Bench.	Tasks	Lat.
Dct1	6	91	**Dct8**	110	147	**MergeSort**	82	87
Dct2	9	153	**Dct9**	141	125	**Fft**	96	253
Dct3	29	130	**Dct10**	146	190	**BeamFormer**	180	159
Dct4	53	106	**InsertionSort**	14	64	**MatrixMult**	200	1087
Dct5	55	170	**RadixSort**	31	138			
Dct6	60	177	**Comp.Count**	49	106			
Dct7	62	174	**JpegDecoder**	50	516			

Figure 3.4 presents the tightness of the *is*WCET over the WCET. For three benchmarks there is no exclusion of interference as there was no interference in the first place (the reason for which is explained the results , below). Since those benchmarks cannot possibly be improved by subsequent steps of the method, they will be omitted from future results.

As a reference of the size of the applications of the benchmark, in Table 3.3 we list their latency according to the over-approximated WCET estimations δ^{over}.

3.3.2 Evaluation Results

In Figure 3.4, the efficiency of the *is*WCET approach is illustrated over the set of benchmarks. We measure the efficiency of *is*WCET, as the percentage of interference the *is*WCET method was able to exclude from the original over-approximated WCET δ^{over}, when the deployment D was not known, to the *is*WCET estimations δ_{ι_D}, when the deployment was provided.

The first observation is that for three benchmarks, i.e. DCT1, DCT2 and DCT4, the *is*WCET method was not able to exclude any interference. For benchmarks DCT1 and DCT2 there was no interference at all, as the amount of tasks was small (<10) due to the low parallelisation considered. This resulted in deployment solutions where the tasks are executed sequentially, thus having no interference. On the other hand, while DCT4 exhibits interference, the deployment process was unable to find an assignment that could reduce interference.

For the remaining fourteen benchmarks, the percentage of reduction ranges from 37% up to 100% with an average of approximately 74%. In fact, for half of those benchmarks, the *is*WCET is able to reduce interference by 80% or more. Since the amount of reduction highly depends on the application under study and the deployment solution, it is challenging to draw concrete conclusions over which applications will benefit most from

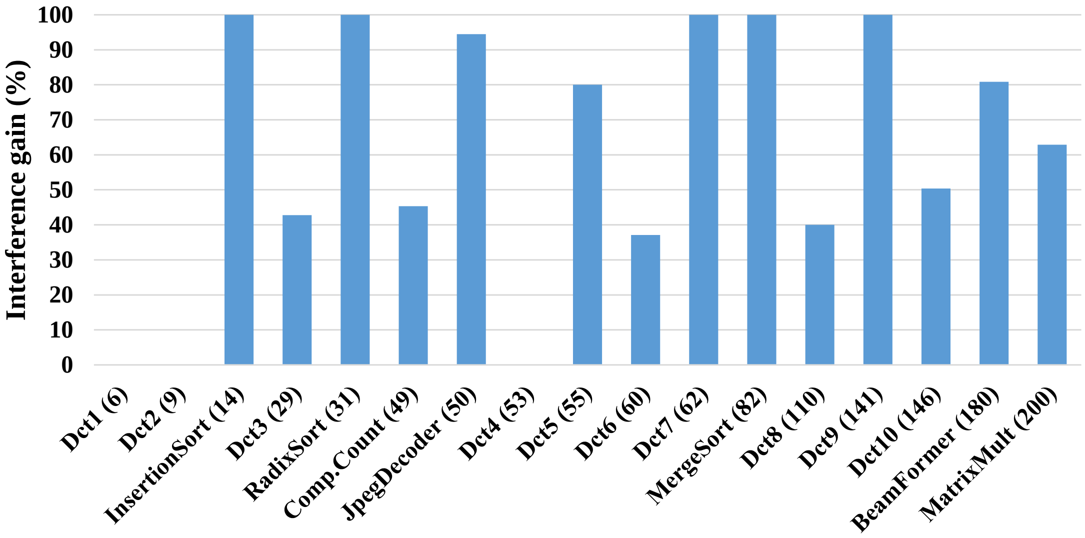


Figure 3.4 – Percentage of spatio-temporally excluded interference on the benchmark set, ordered by increasing number of tasks (in parentheses)

the *is*WCET method. Yet, from the results of this benchmark, we can conclude that, in most cases, the *is*WCET method is able to exclude a significant amount of interference.

Another important measure of efficiency for the *is*WCET method is the percentage of the remaining interference ι_D compared to the overall workload of each benchmark. This can be expressed as the ratio between the the sum of interferences and the sum of *is*WCET estimations for all tasks of the benchmark, i.e. $\sum_{v \in V} \iota_D(v) \Big/ \sum_{v \in V} \delta_{\iota_D}(v)$. We will call that ratio *interference overhead*, as it attributes the amount of time wasted on waiting for resource requests to be served compared to the total time of worst-case execution. For each benchmark, the percentage of interference overhead is illustrated in Figure 3.5.

For benchmarks for which the interference reduction was more than 80% (i.e. InsertionSort, RadixSort, JpegDecoder, DCT5, DCT7, MergeSort and DCT9), the remaining interference overhead is less than 4%. For some benchmarks, the *is*WCET method was able to completely exclude interference: InsertionSort, RadixSort, DCT7, MergeSort and DCT9 in Figure 3.4. For those, the interference overhead (depicted in Figure 3.5) is zero, as expected. On average, the interference overhead of the *is*WCET estimations is less than 10%, which is an important result. An exception to that is the matrix multiplication (MatrixMult) benchmark, which still exhibits an interference overhead of approximately 44%, despite the fact that more than 60% of its interference was excluded. The MatrixMult benchmark is a special case, since it uses simple computational operations, but requires that a significant amount of data shared among tasks. These data cannot be spatially separated (without modifying the source code of the benchmark), such that tasks do not access the same bank. This results in having many parallel tasks accessing the same memory banks, and thus an increased amount of mutual interference.

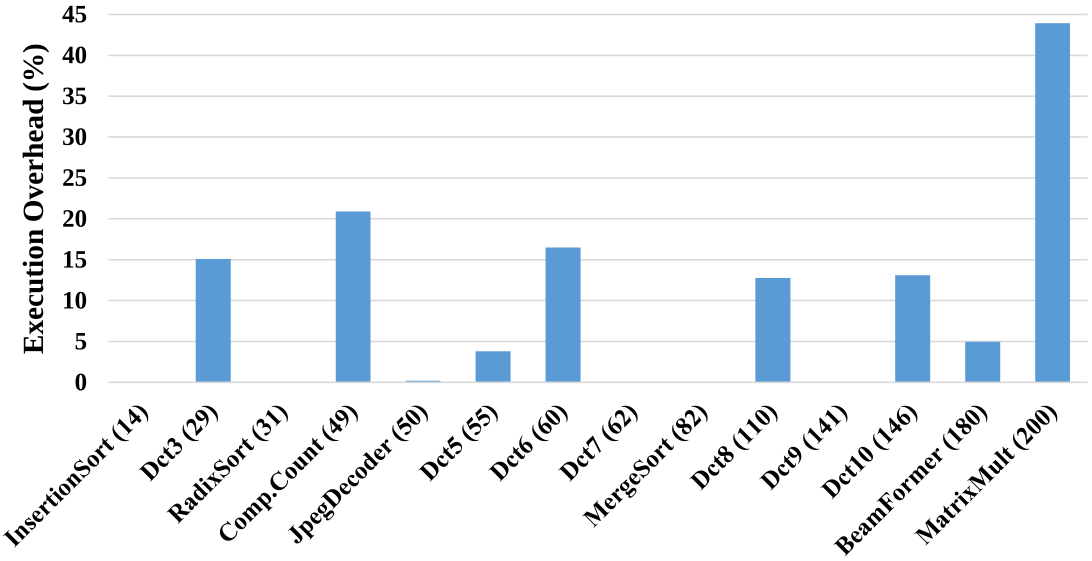
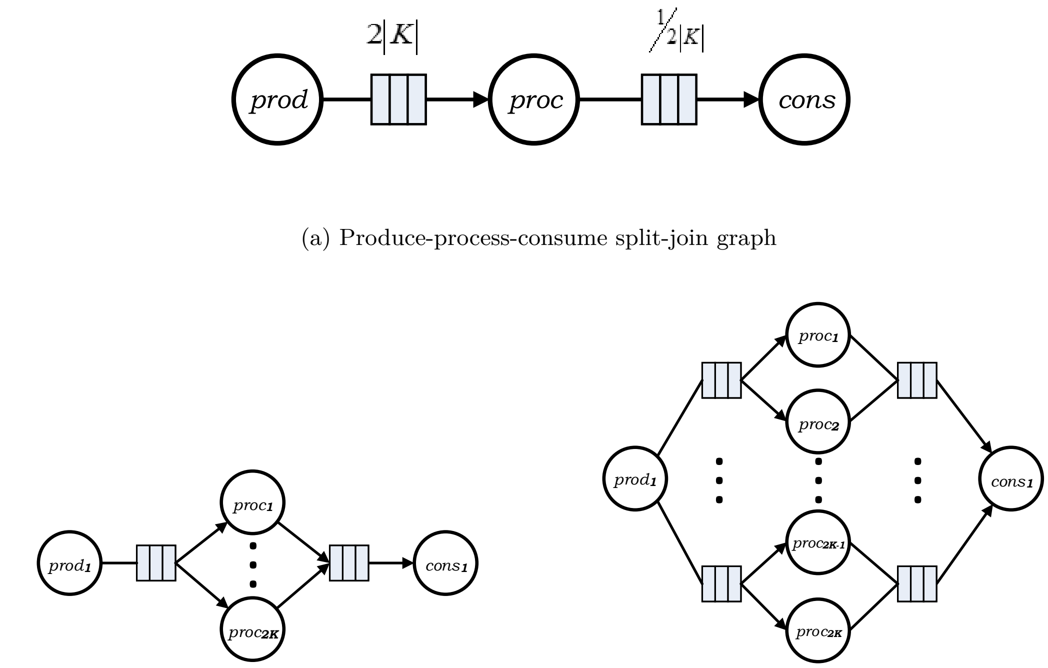


Figure 3.5 – Percentage of the overhead due to interference on the benchmark set, ordered by increasing number of tasks (in parentheses)

To validate this observation, we designed a controlled experiment based on the *producer-process-consumer* paradigm. In this experiment, there are at all times, one producer, one consumer and $2 * |\mathcal{K}|$ $(=32)$ process tasks. The process tasks process 32 data chunks of $1kB$, produced/consumed by the corresponding tasks. The difference in the various instances of the experiment is the number of FIFOs between the process tasks and the producer/consumer tasks, ranging from 2 up to $4 * |\mathcal{K}|$ (illustrated in Figure 3.6).

As the data-parallelisation increases, the FIFOs are split in half and are doubled in number (with a constant capacity of, i.e. $32kB$). Splitting FIFOs enables the deployment process to distribute FIFOs across more memory banks, thus reducing bus and memory interference. Thus, at each increment of the data parallelisation the amount of interference can be approximately halved by carefully mapping FIFOs to memory banks. In turn, this reduces the total latency proportionally (as the WCET of producer/consumer tasks remain constant). This anticipated results are illustrated in Figure 3.7. For data-parallelisation factor equal to twice the number of cores and banks (32), the interference has been already excluded, since the task count exceeds the number of cores and memory banks. Therefore the latency cannot be further improved, and the increased FIFO count brings a small constant overhead, thus the total latency increases slightly.

Finally, it is also important to compare the *is*WCET estimations (δ_{ι_D}) with the initially over-approximated WCET (δ^{over}). This is illustrated in Figure 3.8, where the average task WCET reduction for each benchmark is presented. We can observe that, the *is*WCET estimation are improved by $5-10\%$, on average, for *all* tasks over all benchmarks. This has a significant impact as this implies less worst-case workload and less interference,



(a) Produce-process-consume split-join graph

(b) Task model with data-parallelisation factor 1 and $2|\mathcal{K}|$ process tasks

(c) Task model with data-parallelisation $|\mathcal{K}|$ and $2|\mathcal{K}|$ process tasks

Figure 3.6 – Task model for various data-parallelisation factors, with arrows depicting the input/output FIFOs of tasks

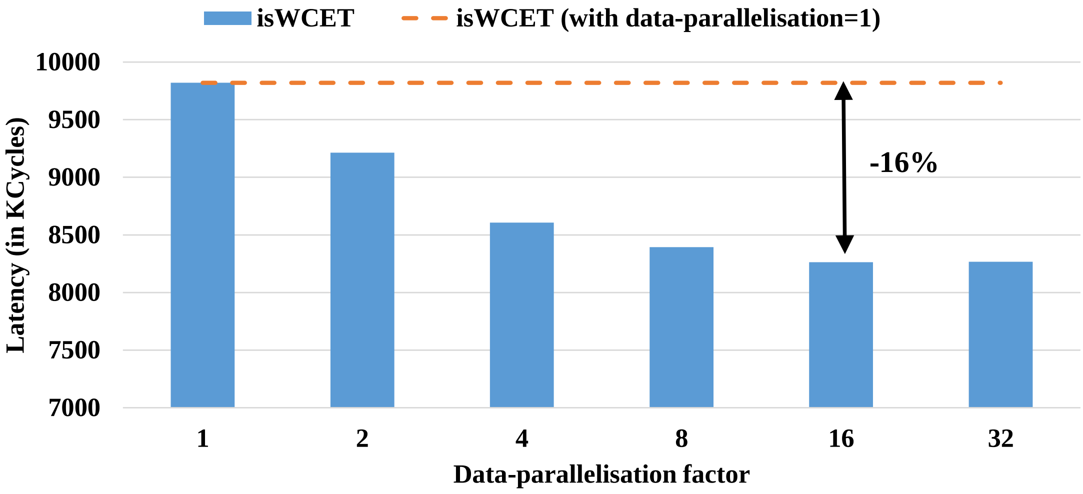


Figure 3.7 – Latency reduction due to accurate WCET estimation accounting for the data-parallelisation factor. Original estimation (dashed line) and corresponding estimation for different values of the data-parallelisation factor, ranging from 1 up to $2 * |\mathcal{K}|$ for a centralised architecture with $|\mathcal{K}| = 16$ cores at 400 MHz and $|\mathcal{M}| = 16$ memory banks

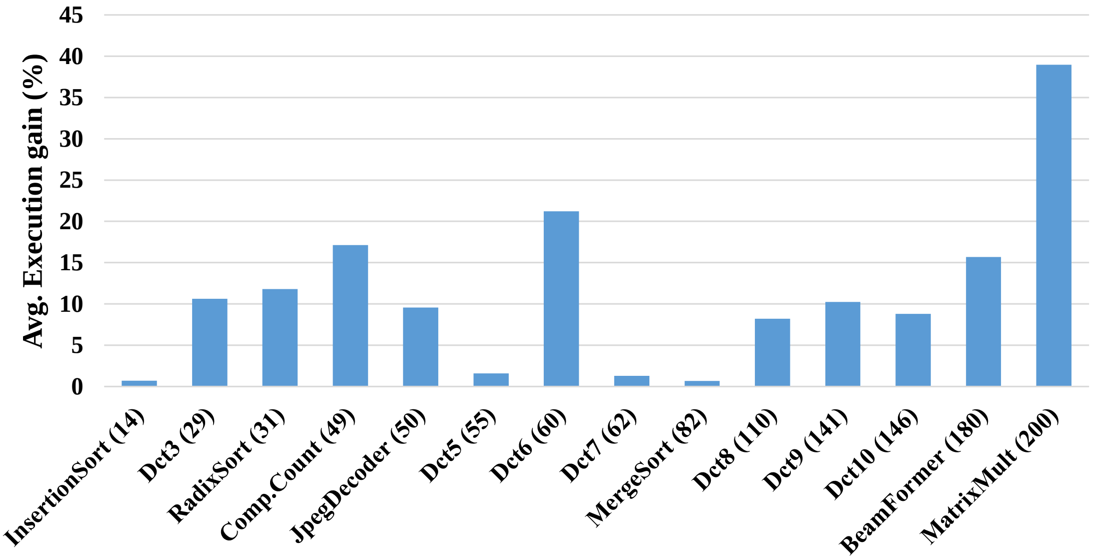


Figure 3.8 – Average percentage reduction of task WCET on the benchmark set, ordered by increasing number of tasks (in parentheses).

thus enabling better utilisation of the resources. Notice that, MatrixMult, which exhibits the highest degree of interference, experiences the most dramatic reduction. In most cases the results show that the count of interfering tasks is lower than the number of cores (i.e. ≤ 16), but for MatrixMult up to 22.

Based on these results, we conclude that in several cases the interference introduced by the parallel execution of tasks can have a significant impact on the WCET of tasks. The proposed *is*WCET method can efficiently exclude interferences, 73% on average, on the majority of the applications of the benchmark. In the cases where the benchmark is small the *is*WCET has limited positive impact, while for highly-parallel applications where data cannot be nicely partitioned, it can exclude the majority of interferences. Yet the interference overhead is not negligible. Thus, based on a controlled experiment, the data-parallelisation factor of an application has to be considered in the interference analysis. In general, the *is*WCET method was able to reduce the WCET of tasks by 10%, on average, up to 37% which is of major importance for the efficiency of hard real-time systems.

3.4 Related work

Interference analysis is far from a new concept, with the first research efforts focusing on resource usage on multi-processor appearing in the early '70s [16, 10] or even earlier. Such efforts focus only in determining the worst-case interference that can happen on a particular architecture, without considering the application and/or the deployment. To the best of the author's knowledge there is exists no other work that couples the interference analysis to the real-time deployment process. In fact, the closest works to the ones proposed are interference-sensitive WCET estimation method [77, 78, 76], where the term *is*WCET was first coined, and the works of the on-going project ARGO [84, 41] where a similar idea of iteratively improving the WCET estimation according to the deployment decisions is advocated and evaluated [90].

The works of Nowotsch, et al., for *is*WCET (first appeared in [77, 78] and extended in [76]) focus COTS platforms where the target architecture is a distributed homogeneous architecture, yet the approach should be extendable to centralised and mixed architectures. The shared resources that are considered are more, as I/O devices, PCI bus, etc. are considered. These works, as opposed to our approaches, enforce resource regulation on a per-task basis in order to ensure that tasks do not generate more interference than permitted. That is, the accesses of task are monitored at runtime and if a task exceeds its allotted resource usage is suspended. A first consideration is that runtime monitoring of resource access can introduce non-negligible overhead to actual execution, which is not addressed in these works. Another, potentially problematic situation, yet a corner case, since mixed-criticality systems are considered is the following. A high-criticality task with tight resource demands on the low-criticality mode to be suspended/slowed-down, while low-criticality tasks run normally, until the criticality mode changes. Still, the approach is of significance and illustrates experimental that a per-task resource-bounded WCET, i.e. *is*WCET, can increase performance significantly.

Of particular interest are interference-free approaches [94, 21, 81]. These approaches require that execution occurs in short phases of local accesses and shared-resources accesses. This can be achieved by reordering/instrumenting the execution binary and properly aligning, in time, the phases of each core such that shared-resource phases do not overlap. This can minimise WCET estimations, as there is no interference, and lead to efficient deployments. Nevertheless, we consider that runtime adaptation to shorter executions of tasks is difficult, as the strict requirement of phase-alignment cannot be violated, and rescheduling at earlier times requires a significant reduction of a single phase. Therefore, such execution models, while efficient in deploying applications for hard real-time, they lack in adaptability.

Following we review some state-of-the-art approaches that are related to WCET and scheduling.

3.4.1 WCET Analysis and Scheduling

We focus on the works that, similarly to us, consider that the WCET of a task is composed of its WCET in isolation, including data fetch and deposit time, with no interferences from other tasks on shared resources plus time delays due to contentions on these resources. There exist several works considering that the WCET of a task is composed of its WCET including data plus time delays due to contentions on these resources. Other papers focus mainly on the WCET estimation of tasks with data dependencies deployed on centralised-[73, 23] and distributed [44] architectures. Works on HW/SW co-design, e.g. [42], can avoid interferences by deciding the amount of resources in the platform, contrary to the commercial hardware that we consider.

In [73] the authors propose an ILP formulation of the task scheduling and mapping problem for multi-core architectures with caches. They consider different communication times for data exchange between the tasks mapped to the same core (e.g. when communication happens through caches) and two different cores (e.g. with the access to shared memory). In [23] author are presenting the upper bound estimation of the WCET also for a memory-centric architecture, similar to ours, by proposing a memory-aware execution to compute the delays due to memory contention. The access to the shared memory is assumed to be realised through Data Memory Access (DMA) units, while the access delays are derived experimentally for different sizes of memory block. These approaches are suitable for architectures with a small number of clusters having simple inter-cluster interconnection network, e.g. TI Keystone II^{TM} [108]. However, the WCET analysis for a multi-processor architectures with multiple clusters interconnected with a NoC such as Kalray MPPA-256 [60] requires more detailed modelling and analysis. There have been several works dedicated to WCET analysis for the tasks with data dependencies deployed onto a multi-processor architecture. In [44] authors are presenting the approach to compute the WCET of tasks running on Kalray MPPA-256 [60] platform by assuming that the maximum number of interfering tasks is equal to the number of processing elements of this cluster, when accessing the shared memory within a cluster, which we experimentally observed that it does not hold in the general case.

In [48], the authors present a comprehensive theory for mixed-criticality scheduling on cluster-based multi-processor architectures with shared resources developed within the CERTAINTY project. To derive a feasible schedule the authors estimate the tasks' worst-case response time (WCRT), which we call WCET[1]. The tasks are scheduled with FTTS mixed-criticality scheduling policy that repeats over a hyper-cycle divided into frames and sub-frames the beginning of which is synchronised among each cores of a cluster. Each sub-frame contains only the tasks of the same criticality level, which ensures that resource contention may happen only among the tasks with the same criticality level. The WCRT for the same criticality level, is composed of a WCET and the total delay due to contention on shared resources. Our model considers no criticality levels but has a detailed representation of the communication mechanism over the NoC. Also, in addition to the arbitration delays when accessing shared memory blocks accounted in [48], we consider the arbitration delays occurring at shared buses. Moreover, we are tightening the WCET by excluding non-interfering tasks, i.e. tasks non-overlapping in

[1] The reason for this conflict of terminology is that often the term WCRT refers to a set of tasks and/or time in which the PE is not blocked and can be exploited

time that share a resource, which is highly complex for the models used in [48].

3.5 Summary

We have presented an accurate estimation of the upper bound for the interference delays of tasks for an application with data dependencies deployed onto a generic architecture. The proposed method produces interference-sensitive WCET estimations, based on the WCET in isolation and according to any provided information (scheduling, mapping etc.) about the deployment.

We experimentally evaluated the *is*WCET method, on Kalray MPPA-256 for the StreamIt benchmark. The proposed *is*WCET method yields an average improvement of the WCET upper bound by 5% to 10% for *all* tasks of all benchmarks. In half of the benchmarks considered, the approach was able to exclude more than 80% of the interferences, thus reducing the interference overhead to less than 10% and improving guaranteed latency up to 46%. Thus, excluding sources of interferences, i.e. using interference-sensitive *is*WCET estimations, can have significant impact on tasks' WCET, which is also reflected in an improvement of overall application latency. This improvement can be achieved without requiring specialised resource partition/regulation techniques, which can undermine runtime performance.

Καιρὸς δ' ἐπὶ πᾶσιν ἄριστος
— Θέογνις (ο Μεγαρεύς)

There is a perfect time for everything
— Theognis of Megara

4 Application Deployment

Given an application, represented with a *Model of Computation* (MoC) such as SDF [68], KPN [49], etc., its deployment on a multi-processor is a constrained multi-criteria optimisation problem [53, 33, 105, 106, 59, 70, 42, 14, 40]. Possible optimisation criteria include latency, throughput, memory usage, etc., while respecting architecture-specific limitations (available memory, number of PEs, communication mechanisms, etc.) and application data-dependencies. Hard real-time systems, additionally, require that given real-time constraints should be met in all possible execution scenarios.

Traditionally, scheduling methods that aim to solve this problem require the a-priori knowledge of the WCET of tasks [34, 25]. For uni-processor systems, that was a reasonable requirement. In multi-processor systems though, for such approaches to be applied, the worst-case has to be assumed about the timing interference due to arbitration of shared hardware resources. This is possible, for some architectures, but the estimated timing interference analysis is performed without any knowledge about how the application is deployed. Such an approach provides run-time *adaptability* by allowing "any" possible task-migration, but it is rather pessimistic as shown in the previous Section and, therefore, can impact latency guarantees. To reduce the pessimism in WCET estimations due to interference, several approaches enforce resource partitioning [102] or resource regulation [36, 66] which, nevertheless, hinder run-time performance as resources cannot be shared or the execution is effectively slowed-down, respectively.

This dissertation proposes to couple the interference estimation with the deployment process in order to cover the ground between scheduling methods and resource partitioning/regulation. Such coupling essentially allows us to explore the trade-off between tighter real-time guarantees and run-time performance. This is achieved by the notion of *is*WCET estimation: depending on the amount of information provided to the interference analysis method, the WCET estimation becomes more accurate, but also more sensitive to deployment changes. This directly affects the number of safe run-time adaptions, such as task rescheduling, migration, etc., available to an online resource manager

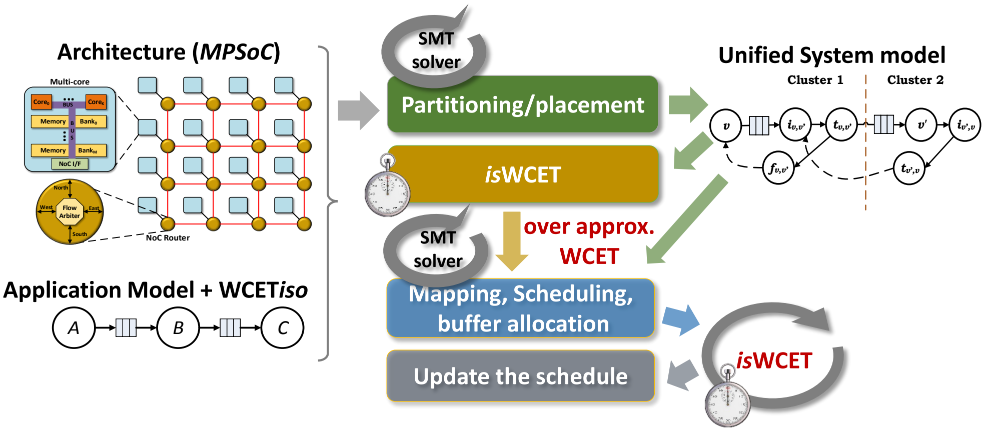


Figure 4.1 – Overview of the deployment approach.

Another way to view the notion of *is*WCET is from the viewpoint of resource regulation. The *is*WCET estimations are effectively resource-limited WCET guarantees expressing that, if a certain amount of interference occurs when a task is executed, then it is guaranteed to finish within the *is*WCET estimation. The key difference with resource regulation techniques lies in the fact that the resource limit of *is*WCET is not predetermined and enforced by hardware or software techniques, but is defined per-task and is left to the deployment process and run-time scheduler to guarantee that this limit will not be violated.

Figure 4.1 illustrates a safe process for deploying an application A (modeled as a split-join graph) to a generic architecture $\mathcal{GA}$ where the deployment is performed in combination with the proposed *is*WCET method. This deployment process is an adaptation of the soft real-time method [105], such that provable real-time guarantees can be provided.

The *safe* and, at the same time, *efficient* deployment of an application A to a generic architecture $\mathcal{GA}$ involves solving the following problems:

- *partitioning/placement* π: assign actors to clusters
- *task-mapping* μ_K: assign tasks (actor firings) to cores
- *scheduling* β: decide the order and begin time of task execution
- *FIFO allocation* ϕ: decide the size of FIFO channels
- *memory-mapping* μ_M: assign FIFO channels to memory banks
- *NoC-routing* ρ: assign NoC routes to NoC transfers

Notice that, for a distributed architecture $\mathcal{DA} = (\mathcal{C}, \{k\}, \mathcal{L}, \mathcal{M}, \mathcal{N})$ the solution π to the partitioning/placement problem is the exact solution of μ_K for the task-mapping problem, since there is only one core per cluster, but in the general case of a generic architecture $\mathcal{GA} = (\mathcal{C}, \mathcal{K}, \mathcal{L}, \mathcal{M}, \mathcal{N})$, the placement solution π is only a partial solution of the task-mapping solution μ_K.

To provide solutions to these optimisation problems, the approach outlined in Figure 4.1 utilises a *Satisfiability Modulo Theory* (SMT) solver. Multi-criteria optimisation problems can be solved using SMT solvers for a bounded cost space, by requesting the SMT for an initial solution within the cost space and, subsequently, iteratively requesting for a solution improved over the current one. When it is not possible possible to improve further the current solution, the solver either replies that the request is unsatisfiable or times-out.

The choice of utilising SMT solvers to solve the optimisation problems, instead of a traditional scheduling approach, is due to the fact that SMT solvers do not have a predetermined behavior. On the contrary, they try to optimise based on an objective function, when used on the previously described way, thus avoiding any probable bias towards interference that other techniques might inherently suffer from. Since, the SMT solver has an arbitrary behavior, it allows us to objectively study the impact of coupling the interference analysis method with the deployment process.

The proposed deployment process, after solving the balance equations for the given split-join graph, partitions the split-join graph (up to the number of clusters $|\mathcal{C}|$) by minimising the amount of data that have to be transferred over the NoC. For each partitioning solution, the partitions are assigned (placed) to clusters by minimising the communication delay (in isolation), which depends on the distance between the communicating clusters. Having acquired a set of partitioning/placement solutions $\{\pi_i\}$, the application model A is transformed into the system model S according to each solution π_i, in order to faithfully model both the computation and communication behaviors of the system. Each system model is then analysed with the *is*WCET interference analysis to acquire safe WCET estimations. Subsequently, the SMT solver solves the task-mapping, scheduling and FIFO allocation optimisation problems all together, trying to minimise the total latency. Using the found solutions, the memory-mapping and NoC routing optimisations, in such away that the number of conflicts in resources is minimised. Aggregating all these solutions, the deployment $D = (\mu_K, \mu_M, \rho, \phi, \beta)$ is generated. As a final step, the system model is analysed once more with the *is*WCET method, but this time, the deployment solution D is provided as well. This results in tighter a *is*WCET, based on which the deployment D is improved via rescheduling.

4.1 Safe Deployment

We shall review the deployment process of Figure 4.1 in a stage-by-stage fashion. For each phase, the models used are defined (if not already), and the corresponding constraint optimisation problem submitted to the SMT solver is presented. Throughout this section, the variables of each optimisation problem are underlined, i.e. "$\underline{x}$" if x is variable and "x" if x is known.

A constrained minimisation problem of the form "*Find* $\underline{X}$ *s.t.* $\min f(\underline{X})$ *subject to* C" can be transformed into a satisfiability problem of the form "*find* $\underline{X_i}$ *s.t.* $\Big(f(\underline{X_i}) < f(X_{i-1})\Big) \wedge C$" iteratively improving the valuation of variables $\underline{X}$. That is, starting from an initial valuation X_0 or upper bound of $f(X)$, iteratively find a valuation of variables $\underline{X_i}$ that is strictly better that X_{i-1} according to optimisation criterion f. To simplify notation, only the constrained version of the minimisation problems, rather than their corresponding satisfiability problems, are presented in the next subsections, but providing information how to acquire the initial valuation or upper bound. The interested readers are referred to [104] for a detailed discussion on the exact formulation of the satisfiability problems.

4.1.1 Partitioning/Placement

In the partitioning step of the deployment process of the approach, the split-join graph is partitioned by balancing the workload among clusters. Since the partitioning cannot be directly linked to latency, that problem is posed a multi-criteria optimisation problem, the two criteria being (i) the number of clusters used and (ii) how balanced a solution is, i.e. the difference of each partition's workload from the average workload.

The workload of an application $A = (\mathcal{U}, \mathcal{E}, \alpha, \delta_{iso}, \sigma)$ is the sum of the durations of all actor firings:

$$W_{\mathcal{U}} = \sum_{u \in \mathcal{U}} \#_u * \delta_{iso}(u) \tag{4.1}$$

where $\#_u$ is the number of firings according to the balance equations. Thus, if x is the number of clusters used, the average workload is $W_{\mathcal{U}}/x$, and $W_{\mathcal{U}_i}$ is the workload of each partition $\mathcal{U}_i$ of the graph. For each value of $x \in [1, |\mathcal{C}|]$, the optimisation criterion to be minimised then is simply expressed as $\sum_{i=1}^{x} \underline{|W_{\mathcal{U}_i}} - W_{\mathcal{U}}/x|$, upper bounded by $2 * W_{\mathcal{U}} * \frac{x-1}{x}$. The upper bound results from assuming the totally imbalanced case, where all the workload is on one cluster, and the rest $x-1$ clusters have no workload at all.

A placement solution π is an assignment of the partitions to clusters. The optimisation criterion used to acquire a placement solution is minimisation of communication delay on the NoC. Clearly, the cost to transfer a specific amount of data depends on the distance among the clusters. The data that are going to be transferred over the NoC are described by the edges of the split-join graph that connect the graph partitions. Thus, the cost function for the placement problem is:

$$\sum_{u \in \mathcal{U}_i, u' \in \mathcal{U}_j} \#_u * r_{in}^{(u,u')} * \sigma(u) * \|\underline{\pi(u)}, \underline{\pi(u')}\|_{\mathcal{L}} \tag{4.2}$$

where $\|\pi(u), \pi(u')\|_{\mathcal{L}}$ is the topological distance according to the NoC links $\mathcal{L}$. Implicitly, the communication cost assumes that the time to traverse the NoC (in absence of interference) is linearly dependent on the distance.

4.1.2 Acquiring the System Model

Having a placement solution π for the given application A, the system model is constructed by adding communication actors that model the timing behavior of NoC transfers. Recall that there are three types of operations that need to be modeled with actors for data transfers over the NoC, namely initialisation, transfer and finalisation. The transformation adds an initialisation actor $i_{(u,u')}$, a transfer actor $t_{(u,u')}$ and a finalisation actor $f_{(u,u')}$ for each data transfer between two actors u, u', placed on different clusters. Each initialisation and finalisation actor have a constant delay, while the duration of the transfer actor depends on the distance among the communicating clusters. Such delays are based on the following constants which are assumed to be known.

Definition 4.1.1 (NoC constants)**.** For a generic architecture model $\mathcal{GA} = (\mathcal{C}, \mathcal{K}, \mathcal{L}, \mathcal{M}, \mathcal{N})$, we define the following timing constants (in cycles):

- $id \in \mathbb{N}_+$ is the initialisation delay of the NoC interface
- $dpp^{\|c,c'\|_{\mathcal{L}}} \in \mathbb{N}_+$ is the delay per packet for transferring data between two clusters c, c' with distance $\|c, c'\|_{\mathcal{L}}$
- $pd \in \mathbb{N}_+$ is the NoC interface polling delay ■

Distributing computation among clusters has a significant consequence for FIFO channels that span across two clusters. From a model perspective that should not cause any issue, but from the perspective of realising the system according to the model, an ambiguity is resolved. A FIFO channel represents a part of the on-chip memory, that an actor can also use as a buffer to store intermediate results. Therefore a FIFO channel spanning across clusters should use either the memory of the source or destination cluster. Yet, the source actor of that FIFO produces data in the source cluster memory, while the destination actor consumes data from the destination cluster memory. The solution is to duplicate the FIFO (one for each side) with the transfer actor transferring (copying) data from one to the other.

This introduces a FIFO protection concern as the source actor has to be aware if the destination FIFO is full or not, before initialising a new data transfer. A way of tackling this concern is by adding a backward notification transfer (message) when the destination actor consumes data, thus informing the source actor regarding the availability of space in the destination FIFO. Thus, in the system model, for each data transfer among actors u, u', five new actors are introduced to model the two-way communication. For the forward data transfer these are, in order of execution, $i_{(u,u')}, t_{(u,u')}, f_{(u,u')}$ for the initialisation, transfer and finalisation respectively. For the backward notification transfer the actors introduced are $i_{(u',u)}, t_{(u',u)}$. Notice the absence of a finalisation actor, since there are no data exchanged and, therefore, this frees no space in any FIFO.

Let $\mathcal{E}_{in}^{A}$ (respectively, $\mathcal{E}_{ex}^{A}$) denote the data-exchange among actors residing in the same cluster (respectively, in different clusters):

$$\mathcal{E}_{in}^{A} = \{(u,u') \in \mathcal{E}^{A} \mid \pi(u) = \pi(u')\}, \quad E_{ex}^{A} = \mathcal{E}^{A} \setminus \mathcal{E}_{in}^{A}$$

Essentially, the transformation of an application A to a system model $\mathcal{S}$ preserves all the intra-cluster data exchanges and replaces the inter-cluster ones, by introducing communication actors and dependencies. It also preserves the possible parallelisation already existing in the application and encompasses possible parallelisation of data transfers.

Additionally, for purposes of memory efficiency in implementation, the semantics of split-join graphs permit the preservation of FIFO places until a subsequent actor is fired, by adding backward edges. For example consider the following sequence of actors $u, i_{(u,u')}, t_{(u,u')}, f_{(u,u')}$ required for a data-transfer over the NoC. Using a FIFO inbetween each of these actors would be inefficient, as the initialisation actor $i_{(u,u')}$ and the finalisation actor $f_{(u,u')}$ do not alter the data to be transferred and simply copy data from their source FIFO to their destination FIFO. The data to be transferred, are considered consumed only when the data transfer is completed, i.e. after the execution the finalisation actor $f_{(u,u')}$, at which point the data are no longer needed.

For this reason for each data transfer, a backward edge is added to the appropriate actors (as illustrated in Figure 4.2). The marking of those edges determines the size of the corresponding FIFOs, and determing it is the objective of the next optimisation step to determine.

Definition 4.1.2 (System model transformation)**.** Given an application model $A = (\mathcal{U}^{A}, \mathcal{E}^{A}, \alpha^{A}, \delta_{iso}^{A}, \sigma^{A})$, a generic architecture $\mathcal{GA} = (\mathcal{C}, \mathcal{K}, \mathcal{L}, \mathcal{M}, \mathcal{N})$ and a placement function π, the *system model* is the annotated split-join graph
$\mathcal{S} = (\mathcal{U}^{A} \cup \mathcal{U}_{Comm}, \mathcal{E}_{in}^{A} \cup \mathcal{E}_{Comm} \cup \mathcal{E}_{\phi}, \alpha, \delta_{iso}, \sigma)$, such that:

- $\mathcal{U}_{Comm} = \{i_{(u,u')}, t_{(u,u')}, f_{(u,u')}, i_{(u',u)}, t_{(u',u)} \mid (u,u') \in \mathcal{E}_{ex}^{A}\}$ is the set of *communication tasks* introduced for the forward and backward transfers among clusters
- $\mathcal{E}_{Comm} = \left\{ \begin{array}{c} (u, i_{(u,u')}) \ , \ (i_{(u,u')}, t_{(u,u')}) \ , \ (t_{(u,u')}, f_{(u,u')}) \ , \ (t_{(u,u')}, u') \ , \\ (u', i_{(u',u)}) \ , \ (i_{(u',u)}, (t_{(u',u)}), \end{array} \middle| \ (u,u') \in E_{ex}^{A} \right\}$
is the set of dependencies that are introduced among computation and communication actors such that the operation of the NoC is respected

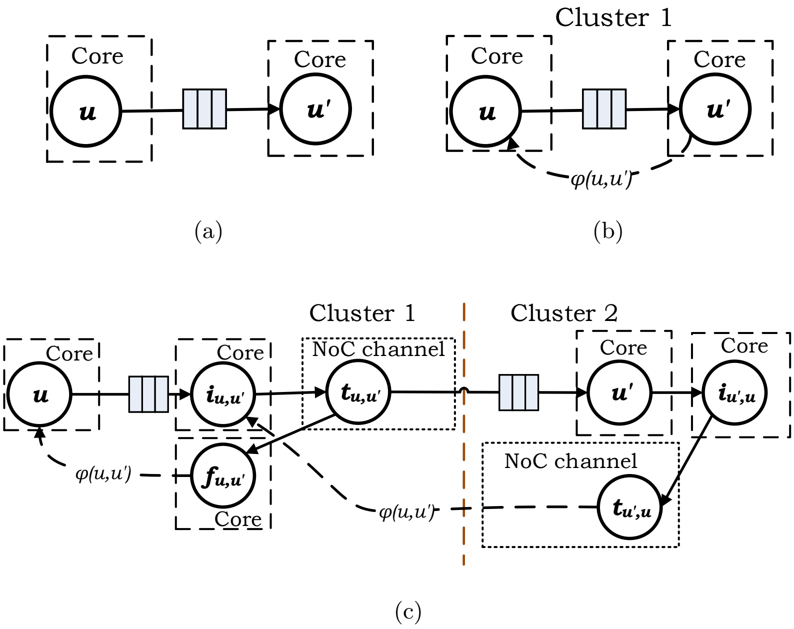


Figure 4.2 – System model for two dependent actors (a), placed on the same (b) and different (c) clusters

Table 4.1 – Extension of functions $\alpha,\delta_{iso},\sigma$ for communication actors

$u \in \mathcal{U}_A$	$\delta_{iso}(.)$
u	$\delta^A_{iso}(u)$

$e \in \mathcal{E}^A_{in}$	$\sigma\,(e)$	$\sigma\left(e^{-1}\right)$	$\alpha(e)$
u, u'	$\sigma^A\left((u,u')\right)$	$\sigma^A\left((u',u)\right)$	$\alpha^A(e)$

$u \in \mathcal{U}_{Comm}$	$\delta_{iso}(.)$
$i_{(u,u')}$ or $i_{(u',u)}$	id
$f_{(u,u')}$ or $f_{(u',u)}$	pd
$t_{(u,u')}$	$dpp^{\Vert\pi(u),\pi(u')\Vert_{\mathcal{L}}} * \sigma_A\left((u,u')\right)$
$t_{(u',u)}$	$dpp^{\Vert\pi(u'),\pi(u)\Vert_{\mathcal{L}}}$

$e \in \mathcal{E}_{Comm}$	$\sigma\,(e)$	$\sigma\left(e^{-1}\right)$	$\alpha(e)$
$u, i_{(u,u')}$	$\sigma^A\left((u,u')\right)$	$\sigma^A\left((u,u')\right)$	1
$i_{(u,u')}, t_{(u,u')}$	0	0	$\vert\mathcal{N}\vert/\alpha^A(e)$
$t_{(u,u')}, f_{(u,u')}$	0	0	$1/\vert\mathcal{N}\vert$
$t_{(u,u')}, u'$	$\sigma^A\left((u,u')\right)$	$\sigma^A\left((u',u)\right)$	$\alpha^A(e)/\vert\mathcal{N}\vert$
$u', i_{(u',u)}$	0	0	1
$i_{(u',u)}, t_{(u',u)}$	0	0	1

- $\mathcal{E}_{\phi} = \{(u', u) \mid (u, u') \in \mathcal{E}^{A}_{in}\} \cup \{(f_{(u,u')}, u)\ ,\ (t_{(u',u)}, i_{(u,u')}) \mid (u, u') \in \mathcal{E}^{A}_{ex}\}$ is the set of backward channels that protect from FIFO overflows, the marking of which shall define the size of FIFOs

- $\alpha, \delta_{iso}, \sigma$ are extended versions of $\alpha^{A}, \delta^{A}_{iso}, \sigma^{A}$ for the additional communication actors and their dependencies, as described by Table 4.1 ■

This system model faithfully captures the behavior of the execution, both in terms of computation and communication. Additionally, the system model properly exploits possible parallelisation, as it retains application parallelism, and communication parallelism, as it allows the transfers to be parallelised up to the maximum allowed by the architecture. This is achieved by allowing each transfer actors to be fired in parallel up to the number of channels $|\mathcal{N}|$.

4.1.3 Scheduling, Task Mapping and FIFO Allocation

The acquired system model is used in this stage to solve the scheduling, mapping and FIFO allocation problems. In order to acquire safe solutions, though, the solutions cannot be based on the WCET in isolation δ_{iso}, since the possible interference is not accounted in those. Thus, the system model $\mathcal{S}$ is transformed to its corresponding execution model $EM(E_S) = (V, E)$ upon which the *is*WCET interference analysis generates over-approximated, but safe, WCET estimations δ^{over}.

Using these WCET estimations, the scheduling, task mapping and FIFO allocation are solved altogether. The constrained optimisation, posed to the SMT solver, that provides solutions to these problems aims to reduce latency. Thus the optimisation criterion is:

$$\min_{v \in V}\ \underline{\epsilon(v)} = \underline{\beta(v)}\ +\ \delta^{over}(v) \text{ subject to } C_D$$

where C_D are the deployment constraints, which guarantee that (i) application dependencies are respected (ii) the real-time constraints are met, (iii) there is enough on-chip memory for the chosen size of FIFOs, (iv) tasks are mapped up to the maximum number of cores per cluster. The cost space of this optimisation is bounded by the serial execution of the actors, i.e. $\sum\limits_{u \in \mathcal{U}} \#_u * \delta^{over}(u)$, as exploring solutions with latency larger than the serial execution cannot possibly be beneficial. Given a system model with safe WCET estimations $\mathcal{S} = (\mathcal{U}, \mathcal{E}, \alpha, \delta^{over}, \sigma)$ and its corresponding execution model $EM(E_S) = (V, E)$, the deployment constraints are the conjunction of the following constraints.

- **Application**: The application dependencies require that some tasks are executed before others, due to data dependencies. These are reflected in the dependency relation E of the execution model EM. Thus, each task must finish before its ancestors can start their execution:

$$\bigwedge_{(v,v')\in E^*} \underline{\epsilon(v)} \leq \underline{\beta(v')}$$

- **Real-time constraints**: Real-time constraints enforce that each actor $u \in \mathcal{U}$ must finish before its respective deadline d_u. This is equivalently expressed as, all tasks $v_u \in V$ of that actor must finish before the deadline d_u:

$$\bigwedge_{v_u\in V} \underline{\epsilon(v_u)} \leq d_u \tag{4.3}$$

- **Task mapping constraints**: Mapping tasks to cores must respect the actor placement π acquired in the previous stage and enforce exclusive use of cores, that is tasks cannot be executed at the same time on the same core. Let the cores of the generic architecture $\mathcal{GA}$ be enumerated in the interval $[1, |\mathcal{C}| * |\mathcal{K}|]$ with the first $|\mathcal{K}|$ belonging to first cluster, etc. Thus, a task v_u of actor u should be mapped onto a core belonging to the cluster where u is placed, i.e.:

$$\bigwedge_{v_u\in V} \pi(u) * |\mathcal{K}| \leq \underline{\mu_M(v)} < (\pi(u)+1) * |\mathcal{K}|$$

  Additionally, tasks mapped in the same core cannot possibly be executed at the same time. That is, for any two tasks mapped on the core, one should finish its execution before the other starts:

$$\bigwedge_{v\neq v'\in V} \underline{\mu_M(v)} = \underline{\mu_M(v')} \Rightarrow \underline{\epsilon(v)} \leq \underline{\beta(v')} \vee \underline{\epsilon(v')} \leq \underline{\beta(v)}$$

- **FIFO constraints** The size of FIFOs has a direct impact on the optimisation problem, as it affects the number of possible solutions. The first necessary condition is that the total size of FIFOs should not exceed the total cluster memory:

$$\bigwedge_{c\in\mathcal{C}} \sum_{e\in\mathcal{E}_c^A} \underline{\phi(e)} * tk_e \leq |\mathcal{M}| * \mathcal{M}_{Size}$$

  where tk_e is the size of a single token for FIFO e and $\mathcal{M}_{Size}$ is the size of a memory bank. Additionally, FIFOs should be protected from overflows and underflows, which also affects possible scheduling solution. Recall that the execution model already contains scheduling dependencies that protect from underflows. Overflow protection dictates that an actor should not fire, if there is not enough space in one of its output FIFOs. Knowing that actors produce and consume ordered streams of data tokens, each at a constant rate, the amount of tokens present in the FIFO can be statically known. For two dependent actors u, u', the amount of tokens is expressed as the difference of produced and consumed tokens between the i^{th} firing of actor u and the j^{th} firing of actor u', i.e. $i * r_{out}^{(u,u')} - j * r_{in}^{(u,u')}$. Thus, if the i^{th}

firing of actor u starts before the j^{th} firing of actor u' ends, the amount of tokens should be at most the size of the FIFO. Therefore, FIFO overflow protection is enforced with the following constraint:

$$\bigwedge_{e=(u,u')\in\mathcal{E}} \bigwedge_{v_u^i, v_u^j \in V} \underline{\beta(v_u^i)} < \underline{\epsilon(v_u^j)} \Rightarrow i * r_{out}^{(u,u')} - (j-1) * r_{in}^{(u,u')} \leq \underline{\phi(e)} \tag{4.4}$$

The solution of this constrained optimisation problem provides a deployment with a guaranteed latency of $\max_{v\in V}\left(\beta(v)+\delta^{over}(v)\right)$. The acquired valuations of the task mapping μ_K, scheduling β and FIFO allocation ϕ functions are part of the deployment solution $D = (\mu_K, \mu_M, \rho, \phi, \beta)$, based on which the memory mapping and NoC routing are solved.

4.1.4 Memory Mapping, NoC Routing

Having a solution for the task scheduling β, the memory mapping optimisation problem consists in find an assignment of FIFOs to memory banks, such that the interference is minimised. Knowledge of the arbitration policies with which the requests of a task will be arbitrated can be exploited to define accurately the optimisation problem. However, to enhance the applicability of the method, we cast the optimisation problem as a minimisation problem of number of conflicts. Notice that, while this is a very reasonable heuristic, an optimal solution of the minimisation of the number of conflicts is not necessarily optimal in terms of interference.

The number of possible conflicts among two non-dependent actors u, u' if one of their FIFOs is mapped on the same memory bank, equals the number of requests of their tasks, but only for those tasks that overlap in time according to the scheduling function β. For example, if u, u' each have only one FIFO e, e', respectively, mapped to the same memory bank, this is formally expressed as:

$$overlap\left(u,u'\right) * \min\left(\sigma\left(e\right), \sigma\left(e'\right)\right)$$

where $overlap\left(u,u'\right)$ is the extension of the $overlap$ predicate for actors, evaluating to the number firings that overlap in time according to β. For dependent actors, i.e. $(u,u') \in \mathcal{E}$, changing the mapping of their in between shared FIFO cannot possibly alter the number of conflicts, or amount of interference.

The memory optimisation problem can be solved on a cluster basis, as the memory mapping of FIFOs on other clusters does not affect the interference occurring on a cluster. Thus, the objective function for the memory optimisation problem is defined as finding a valuation of the mapping function μ_M for the FIFOs $\mathcal{E}_c^A$ of cluster c:

$$\begin{aligned}
&\sum_{\substack{(u_1,u_2)=e\\(u_3,u_4)=e'}\in\mathcal{E}_c^A} \left(\underline{\mu_M(e)} == \underline{\mu_M(e')}\right) * \sum_{\substack{u,\in\{u_1,u_2\}\\u'\in\{u_3,u_4\}}} overlap\left(u,u'\right) * \min\left(\sigma\left(e\right),\sigma\left(e'\right)\right)\\
&\text{subject to } \forall m\in\mathcal{M} \sum_{\underline{\mu_M(e)}=m} \phi\left(e\right) * tk_e \leq \mathcal{M}_{Size}
\end{aligned} \tag{4.5}$$

where tk_e is the size of a single token for FIFO e and $\mathcal{M}_{Size}$ is the size of a memory bank, in words. Notice that this formulation will provide an optimal solution for actors that issue equal numbers of requests. For the general case of unequal requests, the second sum of the minimisation problem is replaced by its expanded form, properly accounting for the different number of requests:

$$\begin{aligned}
&overlap\left(u_1,u_3\right) * \min\left(\sigma\left(e\right),\sigma\left(e'\right)\right)+\\
&overlap\left(u_1,u_4\right) * \min\left(\sigma\left(e\right),\sigma\left(e'^{-1}\right)\right)+\\
&overlap\left(u_2,u_3\right) * \min\left(\sigma\left(e^{-1}\right),\sigma\left(e'\right)\right)+\\
&overlap\left(u_2,u_4\right) * \min\left(\sigma\left(e^{-1}\right),\sigma\left(e'^{-1}\right)\right)
\end{aligned}$$

where $e^{-1}=(u_2,u_1)$ denotes the inverse edge of edge $e=(u_1,u_2)$.

In all cases, the information regarding the number of overlaps, of requests etc. is already known at this stage of the deployment process, and the only variables are the mapping of clusters FIFOs to memory banks.

Similarly, the routing problem is defined as a minimisation problem of the number of conflicts. Following the same convention, the number of possible conflicts among two transfer actors t,t' on a NoC router is number of requests of their tasks, but only for those tasks that overlap. Thus, the objective function for the routing problem is:

$$\sum_{\substack{(t,u)\\(t',u')}\in\mathcal{E}} \left(\underline{\rho(t)}\cap\underline{\rho(t')}\neq\emptyset\right) * overlap\left(t,t'\right) * \min\left(\sigma\left(t,u\right),\sigma\left(t',u'\right)\right) \tag{4.6}$$

Notice that according to the assumed NoC model, two flows that traverse some or all common routers, but in opposite directions, do not interfere with each other, since the NoC routers arbitrate flows towards different directions separately.

4.1.5 Deployment tightening

Having a constructed a safe (see proof in Section 4.1.6) deployment solution $D=(\mu_K,\mu_M,\rho,\beta,\epsilon)$ for a system model $\mathcal{S}=(\mathcal{U},\mathcal{E}\alpha,\delta^{over},\sigma)$, the WCET estimations can be improved using the interference analysis of Chapter 3. Providing the deployment information D to the *is*WCET method results in the tighter interference estimations ι_D and the interference-sensitive WCET:

$$\forall v \in V : \delta_{\iota_D}(v) = \delta_{iso}(v) + \iota_D(v)$$

Replacing the original WCET estimations δ^{over} with the newly acquired *is*WCET results in the interference-sensitive system model $\mathcal{S}_D = (\mathcal{U}, \mathcal{E}\alpha, \delta_{\iota_D}, \sigma)$. Essentially, this means that the interference-sensitive system model and corresponding execution model are faithful only for deployment the given deployment D or for deployments D' where any task v experiences at most $\iota_D(v)$ interference delays. Among these deployments, only those deployments that meet the real-time constraints C_{RT} and can guarantee that FIFOs remain protected, are safe.

In this stage of the deployment process, a safe and improved deployment D' is constructed, based on the already acquired deployment D. This is achieved by modifying the scheduling function β, such that tasks are rescheduled to the soonest possible without introducing new interference.

Given the corresponding execution model $EM(\emptyset) = (V, E)$ for the interference sensitive system model $\mathcal{S}_D = (\mathcal{U}, \mathcal{E}\alpha, \delta_{\iota_D}, \sigma)$, the execution model $EM(\emptyset) = (V, E \cup \emptyset)$ is transformed into the scheduled execution model $EM(E_S) = (V, E \cup E_S)$, by augmenting the dependency relation with scheduling dependencies E_S. The scheduling dependencies enforce that for any two non-overlapping tasks v, v' (i.e. $\beta(v) + \delta_{\iota_D}(v) < \beta(v')$), that share a resource, task v' must be executed only after v has finished:

$$E_S = \left\{(v, v') \in V \mid \beta(v) < \beta(v') \wedge \neg overlap\left(v, v'\right) \wedge share_{r \in R}\left(v, v'\right)\right\} \tag{4.7}$$

This, transformation guarantees that if tasks are rescheduled to earlier times, no new interference will be introduced, as tasks that can interfere on any resource are enforced not overlap (Figure 4.3).

Using the scheduled execution model $EM(E_S) = (V, E \cup E_S)$, the improved deployment $D' = (\mu_K, \mu_M, \rho, \phi, \beta')$ is constructed by rescheduling tasks as early as possible. That is, the scheduling function β' is:

$$\beta'(v) = \max_{(v', v) \in E \cup E_S} \left\{\beta'(v') + \delta_{\iota^D}(v')\right\} \tag{4.8}$$

with $\beta'(v) = 0$ for tasks with no predecessors. The effect of this rescheduling is illustrated in Figure 4.3

4.1.6 Safety

While we proved in the previous chapter that the *is*WCET method is safe, it is important to prove that it has been utilised properly and that the resulting deployment solution is

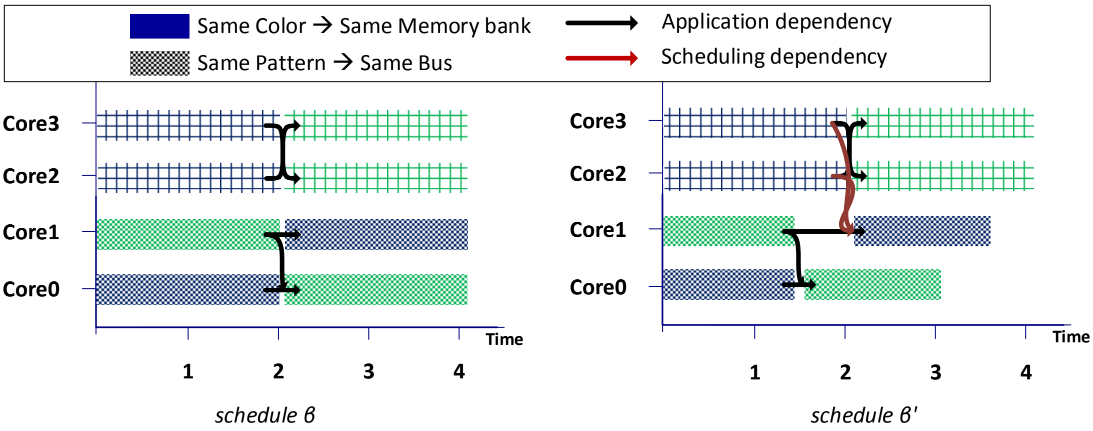


Figure 4.3 – Example of schedule β with data dependencies (black arrows) and β' with scheduling dependencies (red arrows) for one cluster with four cores; same patterned/-colored tasks use the same bus/memory bank, respectively. The green task on Core0 is rescheduled, but the blue task on Core1 is not, as it would introduce new interference to Core2 and Core3.

also safe. A deployment D is safe, according to the Definition 2.3.7, iff all actors finish their firings before their deadlines and the FIFO channels remain protected.

Theorem 4.1.3. *Assuming that the system model $\mathcal{S} = (\mathcal{U}, \mathcal{E}\alpha, \delta^{over}, \sigma)$ contains all the actors that are executed, the deployment D acquired at the "Scheduling, Task Mapping and FIFO allocation" stage is safe.*

Proof. At that stage of the deployment process, since no other actor or task is executed in parallel with the actors of $\mathcal{S}$, the WCET estimations δ^{over} are proven to be safe according to safety of the *is*WCET (Theorem 3.2.7). The fact that actors finish before their respective deadlines is enforced by construction, due to real-time constraints (Equation 4.3) that are supplied to SMT solver. Also, FIFO protection is guaranteed by construction, since FIFOs are protected from underflows according to the transformation of the system model $\mathcal{S}$ to the execution model $EM(\emptyset) = (V, E \cup \emptyset)$ (Definition 2.3.3) and overflows due to the FIFO constraints (Equation 4.4). □

Having proven that the deployment D is safe, we set to prove that the same holds for the tightened deployment D' of the interference-sensitive system model $\mathcal{S}_D = (\mathcal{U}, \mathcal{E}\alpha, \delta_{\iota_D}, \sigma)$.

Theorem 4.1.4. *Given a safe deployment $D = (\mu_K, \mu_M, \rho, \phi, \beta)$ for the system model $\mathcal{S} = (\mathcal{U}, \mathcal{E}\alpha, \delta^{over}, \sigma)$, the deployment $D' = (\mu_K, \mu_M, \rho, \phi, \beta')$ for the interference-sensitive system model $\mathcal{S}_D = (\mathcal{U}, \mathcal{E}\alpha, \delta_{\iota_D}, \sigma)$, acquired at the "Deployment tightening" stage, is safe.*

Proof. According to Theorem 3.2.7 the *is*WCET estimations δ_{ι_D} are also safe, but only if the interference estimations ι_D are not increased for any task. This is guaranteed by construction for the following reason: the additional scheduling dependencies E_S (Equation 4.7) of the scheduled execution model $EM(E_S) = (V, E \cup E_S)$ do not permit additional overlaps of tasks, which share a resource, even if they are scheduled at earlier times. Since only the start times of tasks β' are modified (compared to the safe β), it is not possible to increase the interference estimations ι_D. Thus, rescheduling as soon as possible, according to Equation 4.8, cannot possibly result in any task finishing later compared to the safe deployment D, thus preserving the real-time guarantees □

4.2 Evaluation

4.2.1 Experimental Setup

To experimentally evaluate the coupling of the *is*WCET approach with the deployment process, we use the same setting as Chapter 3, that is 8 distinct applications of the StreamIt [109] benchmark and a JpegDecoder, with Discrete Cosine Transformation (DCT) being modeled in 10 different ways, varying the level of task parallelism.

To acquire safe deployment solutions, we coupled our implementation of the *is*WCET method with the StreamExplorer [104] tool, which provides a set of near-optimal deployment solutions $D = (\mu_K, \mu_M, \rho, \phi, \beta)$ using an SMT solver. Each query to the SMT solver (Z3 [39]) has a time-out of 30 sec. and for each benchmark a global time-out of 6 min.

For the considered benchmark, we chose not impose any deadline, but instead optimise the latency of the application. This forces more tasks to be executed in parallel, thus constructing tightly packed deployments. In these “stressed” deployments, the simultaneous access to resources should be higher, compared to more “relaxed” deployments, thus rendering the improvement of the *is*WCET estimations harder. Therefore, this evaluation of the *is*WCET approach is performed under rather unfavorable conditions, which will outline the benefits of the *is*WCET.

4.2.2 Evaluation Results

Once a deployment solution it acquired, it is of importance to review the latency improvement results in comparison with the results presented in Chapter 3 (for readability and comparison purposes, Figures 3.4, 3.5, 3.8 repeated on the next page). In Figure 4.7, the guaranteed-latency improvement is illustrated for the considered benchmarks. That is, we compare the latency of the whole application computed with the over-approximated WCET (the values of which are in Table 3.3 for reference) to the latency computed with the *is*WCET.

We can observe that the latency improvement varies significantly. Comparing the interference improvement (in Figure 3.4) with the latency improvement (in Figure 4.7),

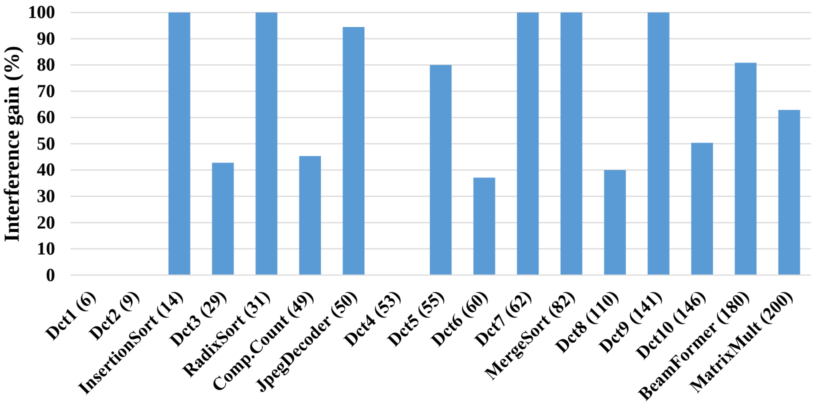


Figure 3.4 – Percentage of spatio-temporally excluded interference per benchmark (repeated from page 45)

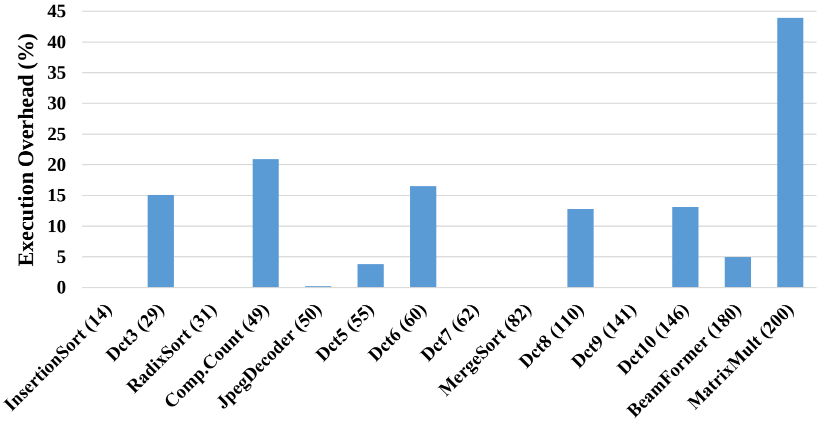


Figure 3.5 – Percentage of the interference overhead per benchmark set (repeated from page 46)

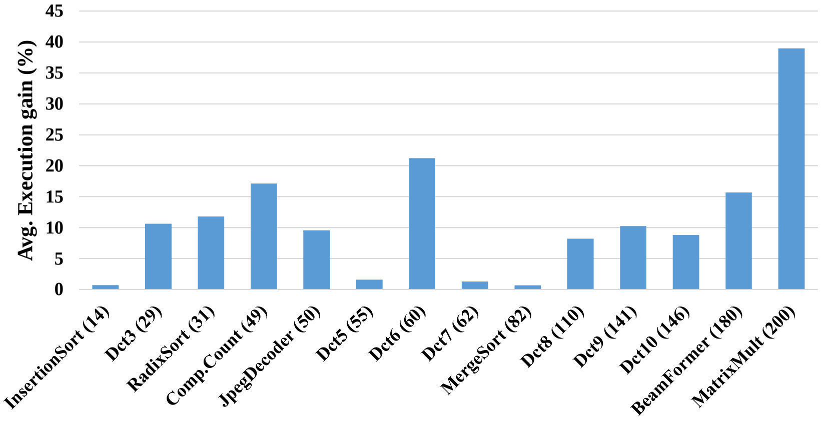


Figure 3.8 – Average percentage reduction of tasks WCET per benchmark (repeated from page 48)

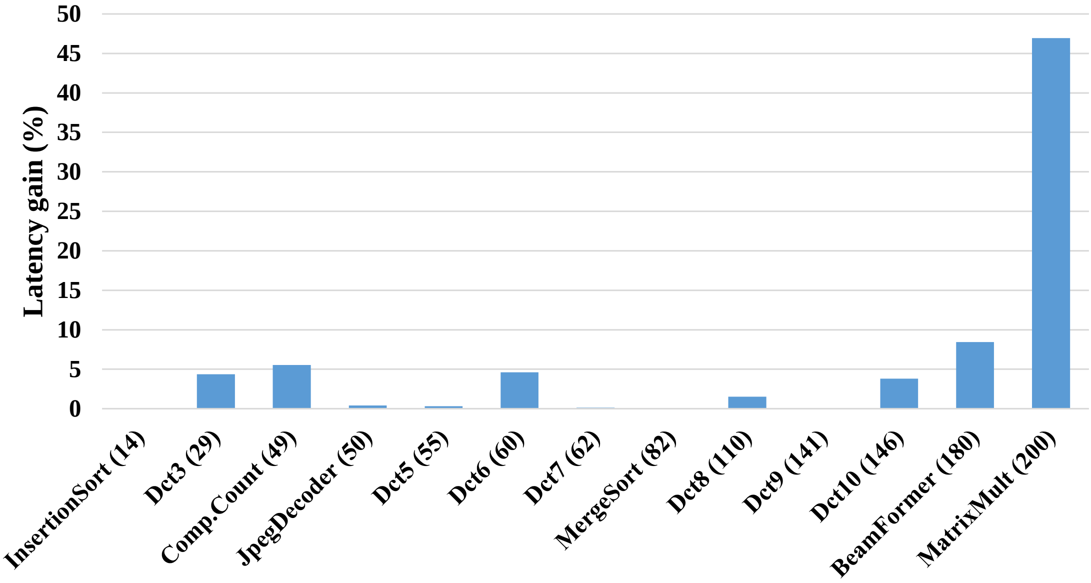


Figure 4.7 – Percentage of guaranteed latency improvement due to spatio-temporal exclusion.

we notice that while there was noticeable reduction of interference, the latency guarantees are not as greatly improved. Especially, for some cases, e.g. MergeSort, even though all the interference is excluded the latency improvement is less than 1%. This is explained by the fact that interfering tasks are not in the critical path of the schedule, i.e. any increase of that tasks execution time does not increase the total latency.

Similarly, for all benchmarks, the latency improvement is slightly lower than the average WCET improvement. This is expected since the WCET improvement (in absolute values) can have at most a linear effect on the overall latency, if those tasks are in the schedule's critical path. This is an important finding as it implies that a significant amount of interference may have no impact on the total guaranteed latency, yet it is still important when considering intermediate deadlines.

Nonetheless, we achieve an average latency-guarantee improvement of 5%, up to 46%, which is significant when providing latency guarantees, with applications, where data cannot be partitioned (MatrixMult), benefiting the most in terms of latency. Also, in [100] we showed that the derived solutions provide guaranteed latency similar to average case solutions of [105], requiring at most 150 KBytes more memory.

4.3 Related work

The latest tendency of migration from single core to multi-processor architectures has raised a question of optimal use of multiple shared resources, e.g. processing cores, shared memories, buses and NoCs. Thus, the optimal deployment of an application, which may have data dependencies, onto a multi-processor platform is regarded as a multi-criteria optimisation problem [33, 105, 106, 59, 70].

Typical approaches from Scheduling Theory [82, 34, 25], although higly valuable, do not cover the full deployment of an application onto a multi-processor architecture with real-time constraints. Most of the existing works can be classified into two broad categories: those optimising the application mapping, buffer allocations and scheduling assuming that the WCET is given, and those estimating the WCET, based on some given solution for allocation deploypement onto an architecture. We review the former, as the latter was already presented in Chapter 3.

In [33, 105, 106] authors proposed to divide such optimisation problem into several stages. Given an application, a multi-processor architecture model and the extra-functional system specification (e.g. latency constraints, memory available for buffer allocation) the set of constraints describing the overall system with its limitations is built and given to an SMT solver to provide optimal solutions for the application mapping, scheduling and buffer allocation. The approach of [105] focuses on achieving performance and not on providing strong guarantees. For this reason the optimization steps are done based on the average case-execution time (ACET) for the computation tasks. Such an approach is suitable for best-effort systems but does not for hard real-time systems, which require the real-time guarantees.

Following we review some state-of-the-art approaches that are related to the deployement problem. Most of these works assume that the WCET time of tasks is known. Also in various works, there are different assumptions about the allowed migrations, different task priorities, etc. [14, 34]. A known issue in multi-processor architectures regarding WCET, is that tasks experience interference thus having varying WCET. This renders such approaches hard to use, as part of our offline phase and in accordance with an accurate interference-based WCET analysis, as there is no guarantee that a newly acquired solution will be better than any previous one or that the method will eventually converge.

In [30], the authors present an optimal scheduling algorithm, LLREF, for periodic task execution on multi-cores which has bounded overhead. The scheduling is founded on T-L planes, like budget or speed diagrams [31], but assumes no context/switching costs nor bounds their number. Similarly, the authors of [88] present an offline/online optimal algorithm, which schedules by reducing the problem to uniprocessor scheduling and bounds the number of migrations/preemptions. The estimated complexity for the online scheduler is $O(kn^2 \log m)$ for n tasks being executed k times on m PEs. We consider that this overhead is not always covered by the gain of avoiding migrations, as the application size grows. The authors, also do not discuss the space complexity of the approach which we consider important for real-time scheduling on many-core architectures, which have rather limited on-chip memory.

Although, such works are valuable, especially for the online part providing better performance, it is hard to envision how these can be employed to provide stronger real-time guarantees. The main reason is that such algorithms will not only change the mapping and scheduling of tasks, thus affecting their WCET, but rarely quantify the latency gain, thus being unable to provide stronger latency guarantee.

The works of [47, 93, 20, 19] discuss the deployment of data-flow applications on multiprocessor platforms. In [20] the authors present an approach to deploy data-flow applications, modeled as SDF graphs, and based on a novel constraint programming algorithm that explores different levels of parallelism and buffer sizes, acquiring solutions that provide throughput guarantees. In [19] they extend the approach including heuristics for more efficient exploration of the solution space. Similarly in [47], the deployment of SDF graphs is addressed using parallel simulated annealing to explore the solution space. This has a significant impact on the applicability of such methods as the model size grows. Although, we do not address such issue as it is out of scope of our current work, our approach is also parallelisable, by dividing the cost space and submitting parallel queries to multiple SMT solvers. In [52] the authors address the issue of exploring the trade-off between data and pipeline parallelism of data-flow applications and optimising buffer-size. This is achieved by introducing a parameterised SDF graph which explicitly captures both aspects of parallelisation. Such work constitutes an interesting direction for our future work.

4.4 Summary

While many formal models for safety-critical real-time or mixed-criticality systems exist, most of them do not explicitly account for interferences. Instead either the *a-priori* knowledge of WCET is assumed, or interference is largely approximated or even safeguarded using either specialized hardware or sophisticated software and scheduling techniques. The latter is especially true for mixed-criticality systems in order to avoid interference from lower criticality tasks, and for some safety-critical approaches as well.

In this chapter, a deployment approach for generic architectures was presented. The deployment approach efficiently deploys data-streaming applications modelled with a complex MoC, while providing real-time guarantees. The deployment process breaks the cyclic dependency between task WCET estimation and deployment solution, by utilising the *is*WCET method, thus acquiring tight WCET and latency guarantees by excluding interference delays. Our experimental evaluation of the StreamIt benchmark on Kalray MPPA-256, shows that the *is*WCETperforms well even under unfavorable setups and can improve guaranteed latency up to 46%. Also as outlined by our experimental evaluation the amount of interference varies significantly and affects guaranteed latency proportionally, but not linearly in the general case, especially in the case of data-parallelisable applications.

As a result, many theoretical approaches [34, 25] can result in inefficient implementations of real-life systems [41]. On the other end of the spectrum, for safety-critical and mixed-criticality systems, there are approaches that focus on a systems perspective. For example the authors of [64, 76] provide sound scheduling and monitoring techniques for the execution of tasks under the presence of interference. Specifically both propose monitoring techniques to safe-guard the amount of interference a task experiences. Nevertheless, such techniques, operating at a low level, cannot optimise according to the parallelisation factor and can introduce non-negligible interference and overhead in the target system by the constant monitoring of resource usage. Finally, model-based techniques [105, 106] can be fitting for the problem as they can explicitly model parallelism and provide safe deployments and are suitable for safe runtime adaptation. While they can achieve good runtime performance, the fact that do not model the interference on the underlying architecture results in pessimistic guarantees and under-utilized systems.

Τί δϵ σοφώτατον; Χρόνος· ανϵυρίσκϵι γαρ πάντα
— Θαλῆς (ὁ Μιλήσιος)

What is the wisest of all? Time; it reveals everything
— Thales of Miletus

5 Runtime Adaptation

In the previous chapters, we proposed an interference analysis method that produces tight and sensitive WCET estimations, which when coupled with the proposed deployment, construct a safe and tight deployment $D = (\mu_K, \mu_M, \rho, \phi, \beta)$ for the system model $\mathcal{S}_D = (\mathcal{U}, \mathcal{E}\alpha, \delta_{\iota_D}, \sigma)$ and its corresponding scheduled execution model $EM(E_S) = (V, E \cup E_S)$. Yet, these *is*WCET estimations δ_{ι_D} are still over-approximations, as they consist of two worst-case factors, WCET in isolation and worst-case interference:

$$\delta_{\iota_D}(v) = \delta_{iso}(v) + \iota_D(v), \ \forall v \in V$$

It is guaranteed that an execution of the deployment D according to the start times of the schedule function β is safe, even if the worst case happens. In actual execution though, the worst-case rarely happens. Even if the worst-case happens for a single task, it is even more unlikely to happen for all the tasks in a single execution iteration. In fact, it is impossible for any pair of parallel tasks to execute with their *is*WCET. Recall that the *is*WCET method, for safety purposes, accounts their mutual interference delays to both tasks. But if one of the tasks executes with its *is*WCET, it is not possible for the other to also execute with its *is*WCET, as the former suffer all the interference delays and the latter none. Thus, a timed execution of the deployment solution according to the schedule function β guarantees safety, but is likely to be inefficient.

Yet, there are systems where, apart from the hard real-time guarantees, the actual performance matters. For example, consider the *Automatic Braking System* (ABS) in the automotive domain, the flight control system in avionics, computer-assisted surgery in the medical domain or even mixed-criticality systems in general. These types of systems, benefit from a faster execution, compared to timed execution, as the system is more responsive. In the case of mixed-criticality systems, a faster execution of higher-criticality tasks would create more slack for the execution of lower-criticality tasks.

To improve the actual performance of the system for the average case, while still meeting the real-time constraints, a runtime adaptation scheme based on the *actual execution time* (AET) of tasks is imperative. There are two categories of runtime adaptation approaches:

- *static adaptation*: where the partial order of tasks, defined by the schedule function β, and mapping of tasks remains the same, but the begin times change.
- *dynamic adaptation*: where the scheduling order, mapping and/or begin times β of tasks can change in any way.

In all cases of run-time adaptation, it must be proven that any rescheduling/remapping decision will not violate the given real-time constraints and latency guarantees. It must be noted that given set of task V, the schedule function β can be constructed offline for any scheduling algorithm. This is can be achieved by applying the scheduling algorithm to the set of tasks V with their WCET, and construct the schedule function β from the WCET estimation and order of tasks. Thus, static adaptation is applicable for any non-preemptive scheduling algorithm, provided that it is proven that such adaptation will not violate the real-time constraints. Considering dynamic adaptation schemes, there are several variations [14, 40] where only a subset of the changes (scheduling order, mapping) are permitted, thus rendering formal proofs easier to achieve and also reducing the algorithm complexity, and by extension its running time.

Another important aspect in runtime adaptation for multi-processor systems, and generally for scheduling algorithms applied at runtime, is the type of implementation, i.e. *centralised* or *distributed*. Scheduling algorithms that are implemented in a centralised manner take into account the global state of the system and typically are executed on a dedicated core. There are several advantages to centralised implementations, as they are easier to prove their correctness and simpler to implement, compared to their distributed versions. Nevertheless, centralised implementations require a dedicated core, which results in sub-optimal usage of resources. In addition, from the time instance when such a scheduler is invoked, until the moment it decides what the adaptation should be, an amount of time has passed and the global state of the system can, in principle, change. This phenomenon can be a source of inefficiency, as the global state can become obsolete and another adaptation may now be more appropriate.

Scheduling algorithms that can be implemented in a distributed manner, in general, take into account a partial state of the system and/or can perform changes only in a subset of the whole systems. A typical example is *work-stealing* [18, 43] methods, where idle cores steal tasks from busy cores with pending tasks. In this case, the scheduler running on that core can only add workload to its core, and not to others. Distributed implementations distribute the workload of rescheduling onto all the available cores and can potentially operate without requiring the global state of the system. The complexity/efficiency trade-off depends on the implementation, among others.

Both types of implementations raise an important issue when it comes to hard real-time systems. The introduction of the schedulers into the system alters the timing behavior, thus potentially violating the static guarantees. A straightforward way to overcome this issue and preserve safety is to introduce upfront scheduling tasks/actors on the model used to derive the safe deployment, or incorporate the WCET of a single invocation of the scheduler to the WCET of each task. In the case of *is*WCET, though, the execution of both centralised and distributed implementations will introduce interference as they have to access the on-chip memory in order to acquire the state of the system, which also have to be incorporated in the *is*WCET of tasks.

For these reasons, in order to acquire *efficient* and *safe* deployment solutions, such runtime adaptation schemes should have the minimal WCET in isolation and minimal interference. In this chapter, a novel static and distributed runtime adaptation technique is proposed, that has minimal WCET and introduced interference. The technique, based on the derived execution model EM and the deployment solution D, constructs a set of dependencies E_{isRA} among the tasks, which guarantee by construction that no new interference will be introduced when rescheduling. This guarantee renders the technique suitable for deployment solutions based on *is*WCET. During execution, based on the actual execution times of tasks, the distributed schedulers apply self-timed scheduling, i.e. whenever the next task is ready (according to the set of dependencies E) it is executed, which is straightforward to implement.

Prior to the description of the adaption approach, several theoretical results are presented that we use to reason about design choices for the proposed approach and aid in proving its safety.

5.1 Theoretical Results

Scheduling algorithms are a deeply investigated field, as it is a fundamental component of operating systems and has applicability from the smallest to the largest computing system. In systems where real-time guarantees are mandatory, such as safety-/mixed-critical systems, studying the behavior of scheduling algorithms is essential. In particular, timing properties, e.g. robustness, freedom of timing anomalies, etc., of said algorithms must be proved for a system to be verified with respect to the necessary real-time guarantees.

While, hard real-time scheduling algorithms assume the WCET, it is possible that when some tasks take less time to execute, the scheduling decisions can actually lead to subsequent tasks to be postponed. That is, if the scheduling algorithm is applied to the WCET δ, the end times ϵ might not be always greater than the end times ϵ_{act}, when the same algorithm is applied to some reduction $\delta_{act}(v) \leq \delta(v)$ for all tasks v. To make this more clear consider the following example.

Example 5.1.1. Consider seven tasks with precedence constraints, illustrated in Fig-

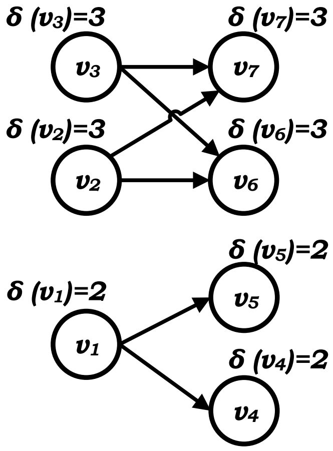
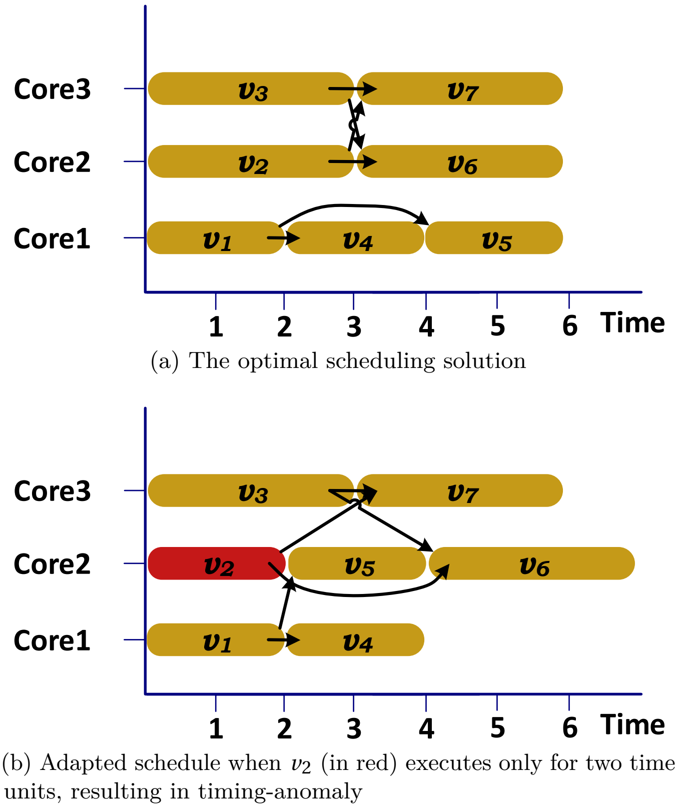


Figure 5.1 – Example task set that can cause a timing anomaly ($\delta(v_i)$ denotes the execution time of that task)



(a) The optimal scheduling solution

(b) Adapted schedule when v_2 (in red) executes only for two time units, resulting in timing-anomaly

Figure 5.2 – Example of a timing-anomaly: seven tasks on three cores and with precedence contraints denoted with arrows.

ure 5.1, that are scheduled on three cores. In Figure 5.2a, a possible schedule is illustrated for these tasks, under the assumption that they will take their WCET to finish execution. Consider the scenario in which task v_2 actually finishes one time unit earlier. This will create a gap, between time instance 2 and 3, in which core 2 is idle. A scheduler could try to reschedule in that gap another tasks which is ready at $t = 2$, e.g. v_5, resulting in v_6 to finish later than its original time, as shown in Figure 5.2b. ■

This is a typical example of what is called *timing* or *scheduling anomaly.* Scheduling algorithms that are free of such anomalies are particularly interesting as they are guaranteed to preserve the safety of deployment solutions.

Recall that since non-preemptive execution is considered the end time of a task v is, by definition, the sum of its start time and its duration, i.e. $\epsilon(v) = \beta(v) + \delta(v)$

Definition 5.1.2 (Scheduling Behavior)**.** Given a set of tasks V, a scheduling function β and the duration function δ, we shall say that a task v precedes another v', denoted as $v \prec_{\epsilon,\beta} v'$, iff v finishes its execution before v' begins (i.e. $\epsilon(v) = \beta(v) + \delta(v) \leq \beta(v')$). A scheduling behavior is the tuple $(\mu_K, \prec_{\epsilon,\beta})$ that consists of the task mapping μ_K and scheduling order $\prec_{\epsilon,\beta}$. ■

Property 5.1.3 (Time-anomaly freedom)**.** A scheduling behavior $(\mu_K, \prec_{\epsilon,\beta})$ is free of *timing anomalies*, with respect to another scheduling behavior $(\mu, \prec_{\epsilon',\beta'})$, iff the end times of the timing behavior of the former (ϵ) are not greater than the end times of the timing behavior of the latter (ϵ'):

$$\epsilon(v) \leq \epsilon'(v) \quad \forall v \in V \tag{5.1}$$

■

Another important property in Scheduling Theory, that aims at efficiency, is *work-conservancy.* Work-conserving algorithms do not allow for a core to remain idle while there is a task ready for execution. This property is fitting to achieve performance for best-effort systems. The intuitive reasoning is, that allowing for a core to remain idle would not contribute positively to the performance of the systems, in the average case.

Yet, such algorithms raise a safety concern, as they can cause timing anomalies and therefore potentially violate the real-time constraints. In fact, we prove that there cannot exist an anomaly-free, work-conserving scheduler, for tasks with dependencies executed in a non-preemptive manner. In order to use such a scheduler, it must be proven that all possible scheduling behaviors, resulting from any possible reduction $\delta^{'}$ of the duration function δ, meet the real-time constraints, which is a combinatorial problem in the general case.

Theorem 5.1.4. *For tasks with dependencies, there is no work-conserving non-preemptive scheduler that is free of time-anomalies.*

Proof. (By counter example) Consider the case presented in Example 5.1.1; after the reduction of v_2, rescheduling v_5 causes a timing anomaly, while not rescheduling violates work-conservancy, as core 2 remains idle while there is a ready task. The same applies for rescheduling v_4. □

Theorem 5.1.5. *Given any set of tasks V with precedence constraints E, let a static scheduler be the one who preserves the task mapping μ_K and the scheduling order $\prec_{\epsilon,\beta}$ for each core, under any reduction $\delta^{'}$ in the duration function δ, i.e. $\forall v \in V, \delta^{'}(v) \leq \delta(v)$. Any non-preemptive static scheduler that schedules tasks as soon as possible is free of timing anomalies.*

Proof. Since tasks do not migrate and are executed according to the scheduling order $\prec_{\epsilon,\beta}$ for each core, it is sufficient to show that tasks on each core are not postponed. On each core, in order for a task v to be postponed, its predecessor must be postponed. As tasks are scheduled as soon as possible and their duration is shorter $\delta^{'}(v) \leq \delta(v)$, it is not possible for any task to be postponed. □

5.2 Interference-sensitive Runtime Adaptation

Following Theorem 5.1.4, in the general case it is impossible to utilise all the timing gaps resulting from shorter execution of tasks. Also, since the proposed runtime adaptation method is aimed at *is*WCET, rather than WCET estimations which account for all possible interference, any memory request performed by the runtime adaptation scheme can potentially introduce interference and violate the already acquired safety guarantees of the deployment D. In order to acquire a safe deployment solution, the additional interference introduced by the runtime adaptation scheme should be a-priori accounted for in the *is*WCET of tasks. In addition, the time that the implementation of the runtime adaptation takes in order to decide what the adaptation should be, must be accounted as well, as it can also be a source of violation of safety.

Thus, there is a trade-off among (i) the complexity of the adaptation scheme, (ii) the amount of information that it requires, (iii) the efficiency of the guaranteed execution of deployment D, and (iv) the actual performance gains. A clear intuitive example of the trade-off, is the comparison of an optimal runtime scheduler with a static one. An optimal runtime scheduler would require the full system state (thus more memory requests) and would be more complex than a static one. Therefore, the optimal runtime scheduler might yield better runtime performance compared to a static one, but worse latency guarantees as it would require more time to execute and generate more interference.

Since this dissertation considers the efficiency of the provided guarantees to be of the highest importance, the proposed approach is a static distributed runtime optimisation technique that has minimal WCET in isolation and memory requests. In addition, as the focus of the dissertation is in *is*WCET-based deployments, the runtime optimisation should guarantee that the real-time constraints, based upon which a safe deployment was constructed, are preserved.

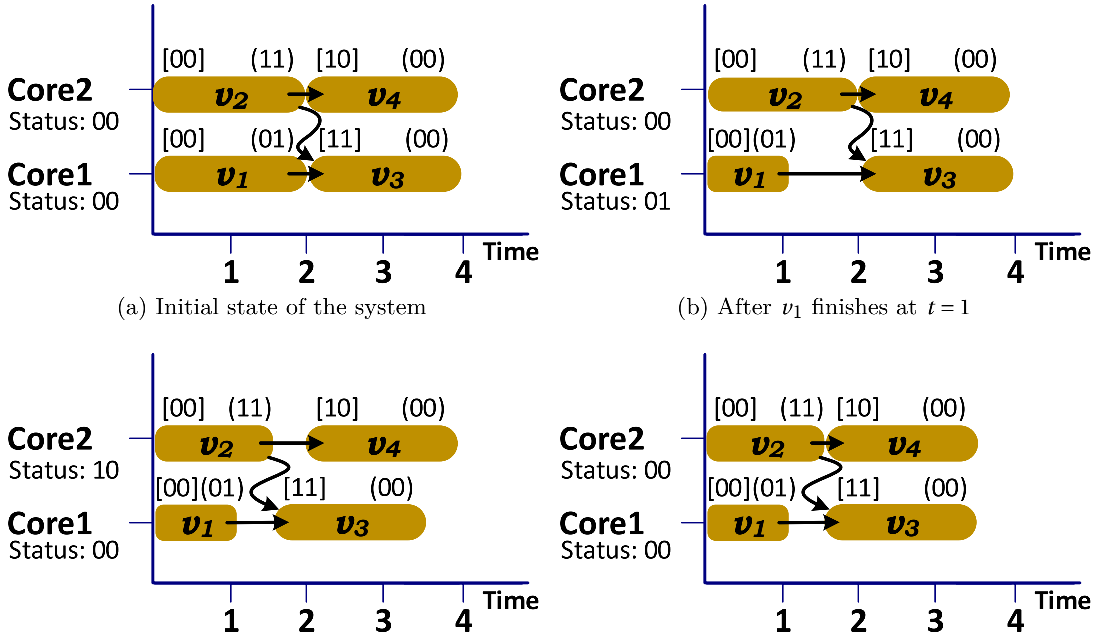
In order to fulfill the latter requirement, a static scheduler that executes the tasks of the execution model $EM(E_S) = (V, E \cup E_S)$ as soon as possible according to the scheduling dependencies E_S would suffice, as the dependency relation E_S guarantees that no new interference would be introduced if tasks are scheduled at earlier times. Nevertheless, the amount of dependencies that the scheduler would have to check can grow arbitrarily large. As a consequence, the WCET and memory requests of such scheduler would grow proportionally to the number of dependencies, which is opposed to the requirement for minimal WCET in isolation and memory requests.

The proposed interference sensitive runtime adaptation approach, for short *is*RA, takes advantage of the spatio-temporal exclusion, but in a much more efficient and predictable manner, so that the WCET in isolation and memory requests are bounded and minimal. Given the set of tasks V to be executed and the safe deployment $D = (\mu_K, \mu_M, \rho, \phi, \beta)$, the proposed *is*RA constructs a set of scheduling dependencies that enforce that every task v' is executed only after the set of tasks $\{v\}$ that precede task v' have finished their execution, with that tasks $\{v\}$ being the last to access a resource that task v' uses:

$$E_{isRA} = \left\{ (v, v') \in V^2 \mid v \prec_{\epsilon,\beta} v' \wedge share_{r \in R}\left(v, v'\right) \wedge \max_{v \in V}(\ \epsilon(v)\ ) \right\} \tag{5.2}$$

Intuitively, these scheduling dependencies preserve the scheduling order and ensure that even if tasks are rescheduled to earlier times, because the AET outperformed the WCET, no task v' will overlap with another task v, that is supposed to finish before v', if they share a resource. In this manner, spatio-temporal exclusion is enforced for tasks that are not supposed to overlap, thus preventing the introduction of additional interference. Notice that according to the E_{isRA} scheduling dependencies the amount of incoming/outgoing dependencies for any task v are bounded by $|\mathcal{K}| * |\mathcal{C}|$, as the number of the latest tasks that finished before v cannot be more than the amount of cores in the architecture.

Having constructed a new scheduled execution model $EM = (V, E_{isRA})$ from the scheduling dependencies E_{isRA}, it is provided to the light-weight distributed monitors, one on each core, for the execution to start. The runtime monitors iteratively execute the set of task V adapting each time to the AET of task. Each monitor is invoked every time a task v finishes, and it reschedules the next task on its core, thus adapting to the AET. In order to preserve safety, the overhead of the monitors is added upfront to the *is*WCET of every task.



(a) Initial state of the system

(b) After v_1 finishes at $t = 1$

(c) At t=1.5, after v_2 finished, $monitor_2$ notifies $monitor_1$ which reschedules $v3$

(d) At t=1.5, $monitor_2$ reschedules v_4

Figure 5.3 – Example of monitor operation for four tasks on two cores. For each task the *ready mask* is in square brackets. The *dependency vector* is in parentheses and is also illustrated with arrows.

5.2.1 Runtime Monitor Operation

The main purpose of monitors is to yield control to the next task when it is *ready* to execute. If not, a monitor must stall until all dependencies are met. In the proposed implementation, each task v holds two bit vectors which are constructed according to the dependency relation E_S:

- the *ready mask* for the incoming edges of v
- the *dependency vector* for the outgoing edges of v

Each bit of these vectors represents the core of the architecture $\mathcal{GA} = (\mathcal{C}, \mathcal{K}, \mathcal{L}, \mathcal{M}, \mathcal{N})$ on which the incoming edge originates from/ the outgoing edge ends in, and is derived from the dependency relation E_{isRA} of the scheduled task model $EM = (V, E_{isRA})$ and the task mapping function μ_K. To facilitate the explanation, it is considered that these vectors always have the same size $|\mathcal{K}| * |\mathcal{C}|$ bits, although for tasks that only have intra-cluster dependencies, $|\mathcal{K}|$ bits are sufficient.

Each monitor also holds a *status* bit vector, of the same size and initialized at 0, tracking from which cores a dependency has been met. Each monitor can write into the other monitors' *status* vectors, either locally or over the NoC.

After a task v' on core k completes, the k-th monitor updates the *status* of all the cores on the *dependency vector* of v', setting their k-th bit, signifying that the dependency from core k has been met. Subsequently, it notifies (e.g. with an interrupt) the monitors of these cores, in case they are waiting for the dependencies of their next task to be met. Finally, the monitor tries to start the next task of the core k; if not ready, it sets the core in idle mode. This process is illustrated in Figures 5.3a, 5.3b and summarised in Algorithm 1.

Algorithm 1: First invocation of a monitor on core k

```
/* Notifies other cores that v' has finished on core $k$ and executes the
next task v when it becomes ready                                        */
input : Task v', status array of all cores;
        (status[$i$][$j$]: the $j$-th status bit of the $i$-th core)
Function invokeMonitor(v',status[ ][ ]):
    v ← v'.next                                      /* 1 on-chip memory read */
    dependencies ← v'.dependencies                   /* 1 on-chip memory read */
    for $i \leftarrow 1$ to $|\mathcal{K}| * |\mathcal{C}|$          /* $|\mathcal{K}| * |\mathcal{C}|$ core-local increments/comparisons */
     do
        if dependencies[$i$] = 1                     /* $|\mathcal{K}| * |\mathcal{C}|$ comparisons */
         then
            status[$i$][$k$] ← 1  /* 1 core-local, $|\mathcal{K}| * |\mathcal{C}| - 1$ on-chip, writes (max) */
            notifyCore($i$)                          /* $|\mathcal{K}| * |\mathcal{C}|$ notifications (max) */
    startTaskWhenReady(v, status[$k$])
```

To start the next task v, the monitor checks if all the required dependencies are met. If not, it enables incoming notifications (e.g. interrupts) and waits for a notification to recheck. When the task is ready, the monitor disables notifications, and resets only the bits of its local *status* indicated by the *ready mask* of task v. (see Figures 5.3c, 5.3d and Algorithm 2). This way, any already-met dependencies of subsequent tasks from cores not indicated by the ready mask, are preserved.

5.2.2 Monitor Overhead

In order to evaluate the overhead of the *is*RA approach, but also to incorporate that overhead into tasks WCET in isolation (as it is required), in Tables 5.1, 5.2 we present the overhead of a monitor in terms of computation and induced interference. Notice that these tables describe the worst-case of computation and induced interference, under the assumption that all tasks have bit vectors of size $|\mathcal{K}| * |\mathcal{C}|$. Since this is true only for transfer tasks, a proper implementation could have even less.

Algorithm 2: Wait until task is ready

```
/* Executes task v when it is ready                                                  */
input : Task v, the status of this core
Function startTaskWhenReady(v,status):
    readyMask ← v.readyMask                              /* 1 on-chip memory read */
    while (status & readyMask) ≠ readyMask    /* $|\mathcal{K}| * |\mathcal{C}|$ AND op./comparisons */
     do
        enableNotifications()                            /* 1 core-local write */
        waitNotify()                          /* $|\mathcal{K}| * |\mathcal{C}|$ wait operation */
    disableNotifications()                               /* 1 core-local write */
    status ← status ⊕ readyMask                          /* 1 on-chip memory write */
    v.execute()
```

Table 5.1 – Monitor computation overhead

Operation Type	**Number of Opearations**
Comparisons	$3 * |\mathcal{K}| * |\mathcal{C}|$
Increments	$|\mathcal{K}| * |\mathcal{C}|$
Assignments (local)	2
Wait Invocation (No-op)	$|\mathcal{K}| * |\mathcal{C}|$

These tables are based on the information provided in the comments of Algorithms 1, 2. Notice in Table 5.1 the wait invocation normally should have zero computation overhead, as typically is a no-op, but for compeletness we list the number of times it is invoked.

5.2.3 Safety

As the proposed method is aimed at hard real-time systems, the safety of the proposed *is*RA method must be formally proven. The deployment $D = (\mu_K, \mu_M, \rho, \phi, \beta)$ that is acquired using the methods proposed in this dissertation, is guaranteed to meet the real-time constraints and that the FIFOs will be protected, only if tasks are executed according to the times provided by the schedule function β.

Table 5.2 – Monitor interference overhead

Opearation Type	**Number of Opearations**
Reads (on-chip)	3
Writes (on-chip)	$|\mathcal{K}| * |\mathcal{C}|$
Notifications	$|\mathcal{K}| * |\mathcal{C}|$

Lemma 5.2.1. For a deployment D, the adaptation scheme *is*RA protects FIFOs from overflows/underflows, under any reduction δ_{act} the runtime adaptation scheme *is*RA if they are protected according to the deployment D.

Proof. In order for a FIFO to overflow/underflow the order of the producing task v and consuming task v' has to be inverted. That is, if according to the safe deployment task v precedes v', i.e. $v \prec_{\epsilon,\beta} v'$, then under any possible actual execution δ_{act} that order must be preserved. According to the definition of the scheduling dependencies E_{isRA} in Equation 5.2, the same scheduling order $\prec_{\epsilon,\beta}$ is enforced by construction. □

Lemma 5.2.2. Given a deployment D, all tasks meet their deadlines under the runtime adaptation scheme *is*RA if their deadlines are met under the timed execution of D.

Proof. Since the execution is non-preemptive, it is sufficient to show that no task will start its execution after its assigned time according to deployment D. Since the adaptation prevents by construction the introduction of new interference, it is guaranteed that the actual execution of any task cannot be greater than its *is*WCET, i.e. $\delta_{act}(v) \leq \delta_{\iota_D}(v), \forall v \in V$. According to the scheduling dependencies of Equation 5.2 and that tasks start as soon as possible, any task v' starts as soon as its predecessors $\{v\}$ have finished, i.e:

$$\beta(v') = \max_{(v,v')\in V} (\ \epsilon(v)\ ) \Rightarrow \beta(v') = \max_{(v,v')\in V} \Big(\beta(v) + \delta_{\iota_D}(v)\Big)$$

Thus, a task can be postponed only if its predecessors are postponed. By backward induction to the tasks with no predecessors, the given is proven. □

Theorem 5.2.3. *The* is*RA adaptation scheme is safe.*

Proof. As an immediate consequence of Lemmata 5.2.1 and 5.2.2. □

5.3 Evaluation

5.3.1 Evaluation Setup

In order to evaluate the proposed runtime adaptation scheme *is*RA, and outline its benefits, we employ the same benchmark used in the previous chapters, that is 8 distinct applications of the StreamIt [109] benchmark and a JpegDecoder, with Discrete Cosine Transformation (DCT), being modeled in 10 different ways. The target platform is a Kalray MPPA-256 chip with SDK version 1.4.1. Later versions of the SDK employ a hypervisor, the source code of which is unavailable, and does not permit full control of the chip. Thus, in order to collect accurate, comparable and reproducible results we opted for version 1.4.1. rather than later versions.

The deployment solution used for the benchmarks was acquired using the methods presented in the previous chapters, with the additional overhead of the *is*RA being added to the WCET in isolation of each task. The deployed solutions are interference-sensitive, that is not all adaptations are safe, but the ones applied by the *is*RAare.

The deployment is described using an XML file, generated by our adapted version of the StreamExplorer [104] tool coupled with the implementation of the *is*WCET method. The deployment XML file is provided to our implementation environment, based on [107], which at the initialisation phase creates the tasks with their bits vectors and allocates the FIFOs on the appropriate memory banks. All FIFOs and their tokens are aligned to memory boundaries, i.e. in memory address in multiples of a word, which affects the number of memory requests. This alignment guarantees that requesting any single-word sized data request will not result into two word memory requests, which could potentially result in a viotation of safety.

Each benchmark application is then executed for 100 iterations, while our implementation measures the latency at the end of each iteration. In order to remain true and accurate regarding the performance results, all executions are performed with the private caches and prefetch buffers of cores disabled. Thus the latency improvement results presented in the next section are indeed benefits of the *is*RA adaptation scheme, and not due to some performance improvement hardware (e.g. caches).

In all 1800 executions (and many more) there was no violation of the latency guarantee.

5.3.2 Evaluation Results

In Figure 5.4, we present a consolidated view of the latency improvements achieved from the deployment stage and the runtime adaptation technique. The height of the column of each benchmark represents the total latency gain (in %), moving from over-approximated WCET, through *is*WCET, to AET.

Recall that the improvement in guaranteed latency achieved during the offline phase varies significantly, for reasons explained in Chapter 4. Nevertheless, we achieve a significant tightening of guaranteed latency of 5%, on average, up to more than 45%. In addition to that, the runtime adaptation technique *is*RA achieves a significant latency reduction, as seen in Figure 5.4. The latency reduction ranges from 3% to 41.5%. The reason for this variation is that the WCET in isolation, and thus the *is*WCET, is still an over-approximation which the runtime adaptation aims to compensate for. This is most apparent in the cases that exhibit no improvement from the offline stage (e.g. DCT1-2, FFT, etc.), which benefit the most from the runtime adaptation. The low latency reduction on some applications (e.g. MergeSort, Beamformer) is justified by the fact that their tasks have a small number of different execution paths, thus their WCET in isolation is quite accurate, and their deployment exhibits no interference or

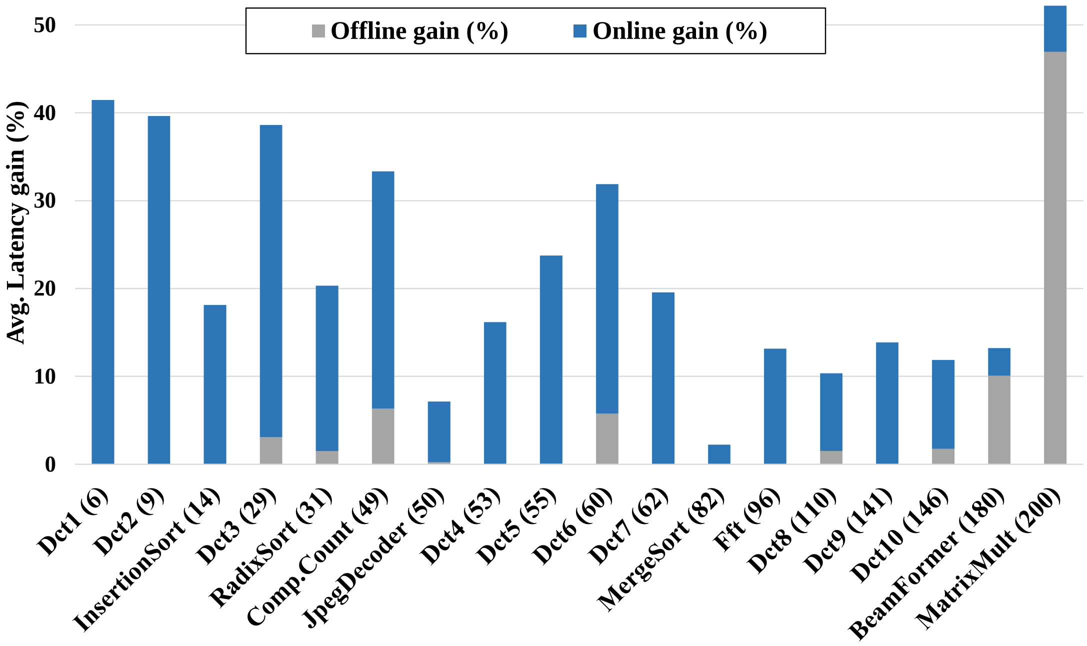


Figure 5.4 – Guaranteed (offline) and runtime (online) latency gain

the interference has been already excluded by the offline phase.

Also, expanding the results of Chapter 3, we also evaluate our runtime adaptation scheme with the *produce-process-consume* application in order to outline the impact of interference as data parallelisation increases in actual executions. In Figure 5.5, we compare the guaranteed latency, acquired through the method of Chapter 4, with the observed latency.

We can observe that data parallelisation, as expected, allows for more parallel FIFO mappings onto memory banks, which improves both the guaranteed and observed latency. The guaranteed latency decreases proportionally to the data-parallelisation factor. The observed latency rapidly decreases for the values of the data-parallelisation factor between 1 and 2, then slightly decreases until value 4, when it starts to slightly increase. This difference is explained by the fact that the *is*WCET analysis expects, for lower values of the data-parallelisation factor, that more memory requests will experience the worst-case delay. In fact, not all memory requests are delayed and not with the worst-case delays. The slight increase, for a data-parallelisation factor >4, of observed latency is either due to higher monitoring overhead or statistical error. The fact that there is only a small difference between the average observed and maximum observed latencies is explained by the fact that, the *consumer* task waits for all the input data, before executing. Thus, the critical path of the application is defined by the slowest pair of tasks on a single core. While, this changes from iteration to iteration, all the tasks have to perform faster in a most iterations, in order to observe a significant difference between the average and

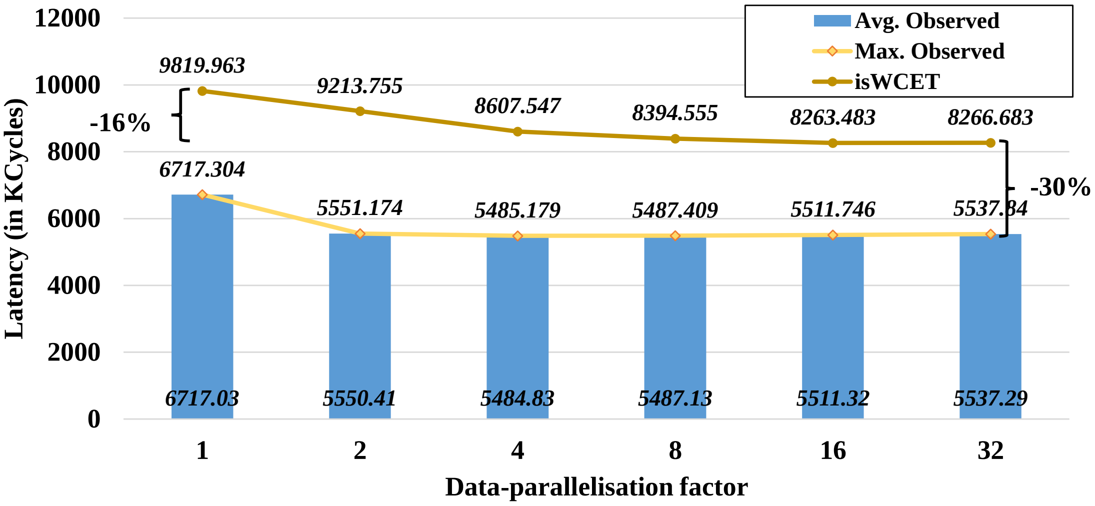


Figure 5.5 – Comparison of the observed latency vs the guaranteed latency of the produce-process-consume application, with 32 process tasks, for different values of data-parallelisation factor deployed on 16 cores at 400MHz.

maximum observed latencies.

These results present interesting findings, as they confirm the need for both offline and runtime optimisations. For the offline optimisation the gain in guaranteed latency can go up to 46%, while the latency gain achieved by the runtime optimisation is approximately 18%, on average, and can reach more that 40%. Their combined gain is 22% on average and can be of more than 50%. Overall, by considering the degree of data parallelisation of the application, we were able to improve the guaranteed latency by 16%. By safely adapting execution, we could compress the runtime by an additional 30%, creating slack that could be used for power-saving modes or the execution of lower-criticality tasks (at the end of execution or using interference monitors [64]).

5.4 Related work

Optimal[1] hard real-time scheduling [14, 40, 34] and runtime adaptation techniques [31, 2] has been the focus of research for a couple decades. For a uniprocessor, *Earliest-Deadline-First* is known to be optimal, but not for multi-processors [82]. The majority of the scheduling techniques assume known WCET [14, 40, 34] and are assumed to be constant. Yet, we have shown that in order to achieve efficient hard real-time systems, this assumption has to be revisited, and the alternative *is*WCET should be adopted.

According to Nowotcsh [76] and to the best of the author's knowledge there is no previous

[1] Optimality, in the context of hard real-time, refers to ability of the scheduler to find a valid schedule that meets the real-time constraints, rather than optimal in terms of latency.

work on runtime suitable for *is*WCET, as it is a relatively new concept (first coined in 2014). Among the literature, we found particularly interesting, the following approaches. The authors of [88] present an offline/online optimal algorithm, which schedules by reducing the problem to uniprocessor scheduling and bounds the number of migrations/preemptions. The estimated complexity for the online scheduler is $O(kn^2 \log m)$ for n tasks being executed k times on m PEs. We consider that this overhead is not always covered by the gain of avoiding migrations, as the application size grows. The space complexity of the approach is not discussed, but seems non-negligible, which we consider important for *is*WCET and real-time scheduling on multi-processor architectures, which have rather limited on-chip memory. The authors of [2], present an alternative approach in solving classic scheduling problems, using timed-automata. By defining the problem, as a timed- automaton they are able to construct adaptation strategies, which is of particular interest for future directions. In [30], the authors present an optimal scheduling algorithm, LLREF, for periodic task execution on multi-cores which has bounded overhead. The scheduling is founded on T-L planes, like budget or speed diagrams [31], but assumes no context/switching costs nor bounds their number.

5.5 Summary

In order to apply runtime adaptation approaches in hard real-time systems, the overhead of the approach must be incorporated in task WCETs, if there is no proof that the method will not violate safety due to its own computational overhead. Incorporating the WCET, and its induced interference due to memory requests, of the runtime to the WCETs of tasks can lead in over-approximated WCETs, thus resulting in inefficient implementations.

In this chapter, a novel adaptation *is*RA scheme is proposes and proven to be safe and having a minimal overhead. The *is*RA scheme is suitable for *is*WCET due to the spatio-temporal isolation mechanism, and can be combined with the works of Nowotsch, et al. [76, 77]. To the best knowledge of the author, there exists no other similar technique that is fit for *is*WCET. The proposed technique is implemented in a distributed fashion and evaluated against the considered benchmark of this dissertation. The results present interesting findings, as they outline the benefits the need for both deployment and runtime optimisations. The deployment process provided a gain in guaranteed latency of 10% up to 46%, while the latency gain achieved by the *is*RA is approximately 18%, on average, and can reach more that 40%. By considering the degree of data parallelisation of the application, the *is*RAcould compress the runtime by an additional 30%, creating slack that could be used for power-saving modes or the execution of lower-criticality tasks (at the end of execution or using interference monitors [64]). The combined gain of all the methods of this dissertation is 22% on average and can be of more than 50%. In all of executions of the benchmark applications, we observed no violation of the guaranteed latency of the deployments constructed with the rest of the methods proposed in this dissertation.

Yet, the *is*RA adaptation technique method is a first attempt to this problem. There are several improvements that are considered as future work. One possible direction, is to dynamically increase adaptability (allow additional overlapping, migrations, etc.) by relaxing the dependency relation as the execution progress and tasks execute faster. This could be possible at the time instance where the additional slack created by previous tasks is larger than the worst-case interference.

6 Conclusions and Future Work

Existing approaches for deploying applications with hard real-time constraints on multiprocessors systems, either assume the worst-case regarding interference delays, due to hardware arbitration of shared resources, or impose resource partitioning/regulation that bounds such delays. The former approaches results in *adaptive*, but inefficient implementations, while the latter in *efficient*, but rather restricted.

The main goal of this dissertation is to stand between the two ends of the spectrum of approaches by enabling to explore the trade-off between adaptability and efficiency. To achieve *efficiency* and retain *adaptability*, we propose that WCET estimation methods and interference analysis methods should be coupled with the deployment process. Specifically, this is achieved by combining single-core WCET methods with interference analysis techniques resulting in *interference-sensitive* WCET (*is*WCET). Such an interference analysis should improve its estimations, thus improving the *is*WCET, as more information regarding the deployment becomes available.

To this end, this dissertation brings several contributions. Initially proposes the *is*WCET method, which based on WCET in isolation, estimates the possible amount of interference, thus producing *is*WCET estimations. The method operates on a generic homogeneous architecture model and an application model with data-dependencies. As more information regarding the deployment is provided, the per-task interference estimation becomes more accurate, but also more sensitive to changes. The second proposal, is a deployment optimisation process, that is coupled with the *is*WCET. The method starting from a MoC (a variant of SDF which explicitly models parallelisation), constructs an abstract system model, based on which various optimisation problems are solved, and its corresponding concrete execution model, on which the interference analysis is performed. The end result of the process, is a tight deployment which when executed in a timely manner is safe. Such deployments are not fully adaptable, as an *is*WCET can be violated if it suffers more interference that specified. Thus, we propose a novel, interference-sensitive, runtime adaptation technique (*is*RA). The *is*RA is suitable for deployments based on *is*WCET

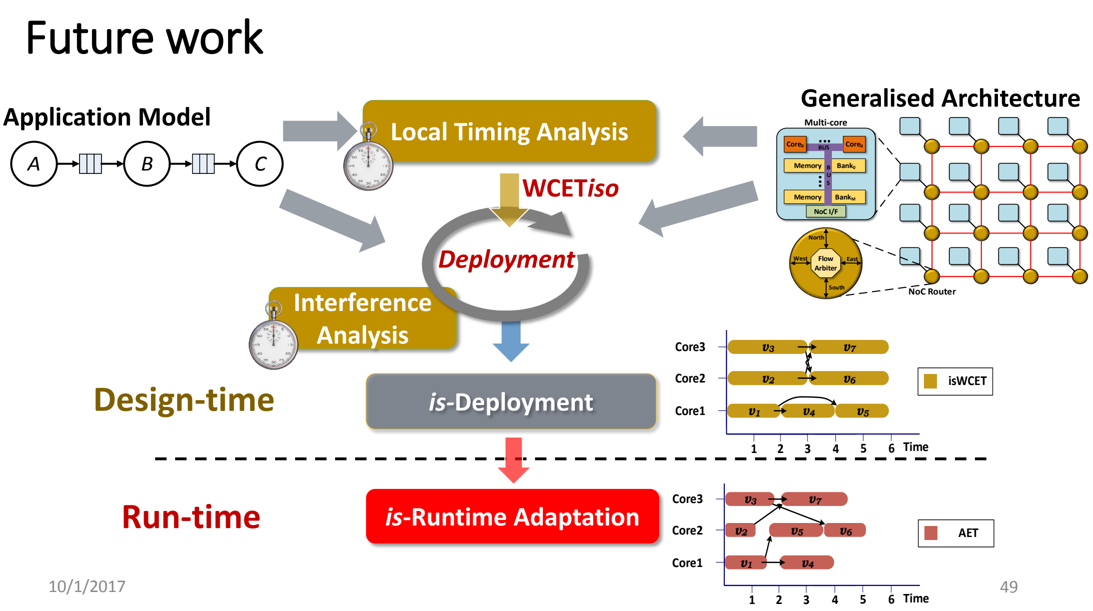


Figure 6.1 – Deployment process, revised for multi-processors.

estimations, as it enforces spatio-temporal exclusion. The overhead of such an approach is minimal, thus suitable for any scale of tasks. All of the approaches were evaluated on an actual multi-processor (Kalray MPPA-256) with real-life applications (StreamIt benchmark), outlining the benefits that each method contributes to the problem of application deployment with real-time guarantees. Experimental results show that our methods improve *is*WCET up to 36%, guaranteed latency up to 46%, runtime performance up to 42%, with a consolidated performance gain of more than 50%. These experimental results indicate that to achieve efficiency it is important to 1) accurately estimate interference delays, 2) consider possible data-parallelisation, as it affects interference, and 3) adapt at runtime with minimal overhead, to preserve safety and efficiency.

Yet, there is more ground to cover, than the already covered by this dissertation. The author envisions an iterative deployment process that improves *is*WCET *until the given real-time*, and possibly thermal/power, constraints are met. Specifically, as illustrated in Figure **??**, such a process is a revised version of traditional one for uni-processors. Starting from an application and architecture model the typical WCET analysis is performed, providing core-local WCET in isolation. The process using the original models and the timing information, iteratively improves the interference estimations and deployment solution until the given constraints are met. At each iteration the solution becomes more static, as more deployment decisions are fixed, thus less adaptive, but more efficient since the estimations are improved. The resulting solution should be an interference-sensitive deployment, which based on the deployment decisions, should permit only a subset of the possible adaptations, in order to preserve safety.

This visionary deployment process, opens several questions for discussion, such as "Which deployment decision should be taken first?". As stated in the introduction of this dissertation, these problems are interdependent and there is no clear justification or reasoning to argue about in which order these problems should be solved and whether the order affects the efficiency of the deployment solution or not. Another important research question is "Can partial solutions to the optimisation problems, provide a safe solution?". Such partial solutions could be of the form "Actor u can be placed in cluster c_1 or c_2" or "Task v should be executed before task v'" without precisely identifying a single deployment solution, but rather a family of safe solutions. This would result in more adaptable system, as more adaptations are permitted to performed at runtime. This fact, in turn, raises another important question "Apart from *is*RA, which runtime adaptation scheme is safe to apply for a given deployment?". While the resulting system is more adaptable, it also would require a more sophisticated runtime adaptation scheme, the overhead of which has also to be taken into account. Despite the system being more adaptable, in principle, efficient runtime adaptation schemes should be devised in order to preserve the safety and efficiency of the deployment solutions.

Such research questions, along with alleviating some of our assumptions, such as different application models, non-preemptive execution, different NoC model, etc., and studying the corresponding systems are left for future work.

Bibliography

[1] aiT: The industry standard for static timing analysis: http://www.absint.com/ait/.

[2] Yasmina Abdeddai, Eugene Asarin, Oded Maler, et al. Scheduling with timed automata. *Theoretical Computer Science*, 354(2):272–300, 2006.

[3] Rajeev Alur and David Dill. Automata for modeling real-time systems. In *International Colloquium on Automata, Languages, and Programming*, pages 322–335. Springer, 1990.

[4] Aravindh Anantaraman, Kiran Seth, Eric Rotenberg, and Frank Mueller. Enforcing safety of real-time schedules on contemporary processors using a virtual simple architecture (visa). In *Real-Time Systems Symposium, 2004. Proceedings. 25th IEEE International*, pages 114–125. IEEE, 2004.

[5] James H Anderson, Sanjoy Baruah, and Björn B Brandenburg. Multicore operating-system support for mixed criticality. In *Proceedings of the Workshop on Mixed Criticality: Roadmap to Evolving UAV Certification*. Citeseer, 2009.

[6] Todd Austin, David Blaauw, Scott Mahlke, Trevor Mudge, Chaitali Chakrabarti, and Wayne Wolf. Mobile supercomputers. *Computer*, 37(5):81–83, 2004.

[7] Hamdi Ayed, Jérôme Ermont, Jean-luc Scharbarg, and Christian Fraboul. Towards a unified approach for worst-case analysis of tilera-like and kalray-like noc architectures. In *Factory Communication Systems (WFCS), 2016 IEEE World Conference on*, pages 1–4. IEEE, 2016.

[8] Sanjoy K Baruah, Liliana Cucu-Grosjean, Roabert I Davis, and Claire Maiza. Mixed criticality on multicore/manycore platforms (dagstuhl seminar 15121). In *Dagstuhl Reports*, volume 5. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2015.

[9] Sanjoy K. Baruah, Gilad Koren, Bhubaneswar Mishra, Arvind Raghunathan, Louis E. Rosier, and Dennis E. Shasha. On-line scheduling in the presence of overload. In *32nd Annual Symposium on Foundations of Computer Science, San Juan, Puerto Rico, 1-4 October 1991*, pages 100–110, 1991.

[10] Forest Baskett and Alan Jay Smith. Interference in multiprocessor computer systems with interleaved memory. *Communications of the ACM*, 19(6):327–334, 1976.

[11] Ananda Basu, Marius Bozga, and Joseph Sifakis. Modeling heterogeneous real-time components in bip. In *Software Engineering and Formal Methods, 2006. SEFM 2006. Fourth IEEE International Conference on*, pages 3–12. Ieee, 2006.

[12] Luca Benini and Giovanni De Micheli. Networks on chips: A new soc paradigm. *computer*, 35(1):70–78, 2002.

[13] Guillem Bernat, Alan Burns, and Albert Liamosi. Weakly hard real-time systems. *IEEE transactions on Computers*, 50(4):308–321, 2001.

[14] Marko Bertogna. *Real-time scheduling analysis for multiprocessor platforms*. PhD thesis, 2008.

[15] Marko Bertogna and Michele Cirinei. Response-time analysis for globally scheduled symmetric multiprocessor platforms. In *Real-Time Systems Symposium, 2007. RTSS 2007. 28th IEEE International*, pages 149–160. IEEE, 2007.

[16] Dileep P Bhandarkar. Analysis of memory interference in multiprocessors. *IEEE Transactions on Computers*, 100(9):897–908, 1975.

[17] G. Bilsen, M. Engels, R. Lauwereins, and J. Peperstraete. Cycle-static dataflow. *IEEE Transactions on Signal Processing*, 44(2):397–408, Feb 1996.

[18] Robert D Blumofe and Charles E Leiserson. Scheduling multithreaded computations by work stealing. *Journal of the ACM (JACM)*, 46(5):720–748, 1999.

[19] Alessio Bonfietti, Luca Benini, Michele Lombardi, and Michela Milano. An efficient and complete approach for throughput-maximal SDF allocation and scheduling on multi-core platforms. In *Proceedings of the Conference on Design, Automation and Test in Europe*. European Design and Automation Association, 2010.

[20] Alessio Bonfietti, Michele Lombardi, Michela Milano, and Luca Benini. Throughput constraint for synchronous data flow graphs. In Willem-Jan van Hoeve and JohnN. Hooker, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, volume 5547 of *Lecture Notes in Computer Science*, pages 26–40. Springer Berlin Heidelberg, 2009.

[21] Frédéric Boniol, Hugues Cassé, Eric Noulard, and Claire Pagetti. Deterministic execution model on cots hardware. *Architecture of Computing Systems–ARCS 2012*, pages 98–110, 2012.

[22] Marius Bozga, Conrado Daws, Oded Maler, Alfredo Olivero, Stavros Tripakis, and Sergio Yovine. Kronos: A model-checking tool for real-time systems. In

International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems, pages 298–302. Springer, 1998.

[23] P. Burgio, A. Marongiu, P. Valente, and M. Bertogna. A memory-centric approach to enable timing-predictability within embedded many-core accelerators. In *Real-Time and Embedded Systems and Technologies (RTEST), 2015 CSI Symposium on*, pages 1–8, Oct 2015.

[24] Alan Burns and Sanjoy Baruah. Timing faults and mixed criticality systems. In *Dependable and Historic Computing*, pages 147–166. Springer, 2011.

[25] Alan Burns and Robert Davis. Mixed criticality systems-a review. *Department of Computer Science, University of York, Tech. Rep*, 2013.

[26] Giorgio C Buttazzo. *Hard real-time computing systems: predictable scheduling algorithms and applications*, volume 24. Springer Science & Business Media, 2011.

[27] John M Calandrino, James H Anderson, and Dan P Baumberger. A hybrid real-time scheduling approach for large-scale multicore platforms. In *Real-time Systems, 2007. ECRTS'07. 19th Euromicro Conference On*, pages 247–258. IEEE, 2007.

[28] Sudipta Chattopadhyay, Abhik Roychoudhury, and Tulika Mitra. Modeling shared cache and bus in multi-cores for timing analysis. In *Proceedings of the 13th international workshop on software & compilers for embedded systems*, page 6. ACM, 2010.

[29] Bo Chen, Chris N Potts, and Gerhard J Woeginger. A review of machine scheduling: Complexity, algorithms and approximability. In *Handbook of combinatorial optimization*, pages 1493–1641. Springer, 1998.

[30] Hyeonjoong Cho, Binoy Ravindran, and E Douglas Jensen. An optimal real-time scheduling algorithm for multiprocessors. In *Real-Time Systems Symposium, 2006. RTSS'06. 27th IEEE International*, pages 101–110. IEEE, 2006.

[31] Jacques Combaz, Jean-Claude Fernandez, Thierry Lepley, and Joseph Sifakis. QoS control for optimality and safety. In *EMSOFT'05*, pages 90–99. ACM, 2005.

[32] Érika Cota, Alexandre de Morais Amory, and Marcelo Soares Lubaszewski. *Reliability, Availability and Serviceability of Networks-on-chip.* Springer Science & Business Media, 2011.

[33] Scott Cotton, Oded Maler, Julien Legriel, and Selma Saidi. Multi-criteria optimization for mapping programs to multi-processors. In *Industrial Embedded Systems (SIES), 2011 6th IEEE International Symposium on*, pages 9–17. IEEE, 2011.

[34] Robert I Davis and Alan Burns. A survey of hard real-time scheduling for multiprocessor systems. *ACM computing surveys (CSUR)*, 43(4):35, 2011.

[35] Benoît Dupont de Dinechin, Renaud Ayrignac, Pierre-Edouard Beaucamps, Patrice Couvert, Benoit Ganne, Pierre Guironnet de Massas, François Jacquet, Samuel Jones, Nicolas Morey Chaisemartin, Frédéric Riss, et al. A clustered manycore processor architecture for embedded and accelerated applications. In *High Performance Extreme Computing Conference (HPEC), 2013 IEEE*, pages 1–6. IEEE, 2013.

[36] Benoît Dupont de Dinechin, Yves Durand, Duco van Amstel, and Alexandre Ghiti. Guaranteed services of the noc of a manycore processor. In *Proceedings of the 2014 International Workshop on Network on Chip Architectures*, NoCArc '14, pages 11–16, New York, NY, USA, 2014. ACM.

[37] Benoît Dupont de Dinechin, Duco van Amstel, Marc Poulhiès, and Guillaume Lager. Time-critical computing on a single-chip massively parallel processor. In *Design, Automation and Test in Europe Conference and Exhibition (DATE), 2014*, pages 1–6. IEEE, European Design and Automation Association, 2014.

[38] Benoît Dupont de Dinechin, Pierre Guironnet de Massas, Guillaume Lager, Clément Léger, Benjamin Orgogozo, Jérôme Reybert, and Thierry Strudel. A distributed run-time environment for the kalray mppa®-256 integrated manycore processor. *Procedia Computer Science*, 18:1654 – 1663, 2013. 2013 International Conference on Computational Science.

[39] Leonardo De Moura and Nikolaj Bjørner. Z3: An efficient smt solver. *Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340, 2008.

[40] Paulo Manuel Baltarejo de Sousa. *Real-Time Scheduling on Multi-core: Theory and Practice.* PhD thesis, Universidade do Porto (Portugal), 2013.

[41] Steven Derrien, Isabelle Puaut, Panayiotis Alefragis, Marcus Bednara, Harald Bucher, Clement David, Yann Debray, Umut Durak, Imen Fassi, Christian Ferdinand, et al. Wcet-aware parallelization of model-based applications for multi-cores: The argo approach. In *2017 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 286–289. IEEE, 2017.

[42] Robert P Dick and Niraj K Jha. Mogac: a multiobjective genetic algorithm for hardware-software cosynthesis of distributed embedded systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 17(10):920–935, 1998.

[43] James Dinan, D Brian Larkins, Ponnuswamy Sadayappan, Sriram Krishnamoorthy, and Jarek Nieplocha. Scalable work stealing. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, page 53. ACM, 2009.

[44] Amira Dkhil, Stephane Louise, and Christine Rochange. "worst-case communication overhead in a many-core based shared-memory model". In *Junior Researcher Workshop on Real-Time Computing, Nice, 16/10/2013-18/10/2013*, pages 53–56, http://www.uva.nl, octobre 2013. University of Amsterdam.

[45] Jianzhong Du and Joseph Y-T Leung. Complexity of scheduling parallel task systems. *SIAM Journal on Discrete Mathematics*, 2(4):473–487, 1989.

[46] Alejandro Duran and Michael Klemm. The intel® many integrated core architecture. In *High Performance Computing and Simulation (HPCS), 2012 International Conference on*, pages 365–366. IEEE, 2012.

[47] François Galea and Renaud Sirdey. A parallel simulated annealing approach for the mapping of large process networks. In *Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2012 IEEE 26th International*, pages 1787–1792. IEEE, 2012.

[48] Georgia Giannopoulou, Nikolay Stoimenov, Pengcheng Huang, Lothar Thiele, and BenoîtDupont de Dinechin. Mixed-criticality scheduling on cluster-based manycores with shared communication and storage resources. *Real-Time Systems*, pages 1–51, 2015.

[49] Kahn Gilles. The semantics of a simple language for parallel programming. *In Information Processing*, 74:471–475, 1974.

[50] Joel Goossens, Sanjoy Baruah, and Shelby Funk. Real-time scheduling on multiprocessors. In *the 10th International Conference on Real-Time System*, 2002.

[51] Mark Harris. Optimizing cuda. *SC07: High Performance Computing With CUDA*, 2007.

[52] Joost PHM Hausmans, Stefan J Geuns, Maarten H Wiggers, and Marco JG Bekooij. Unified dataflow model for the analysis of data and pipeline parallelism, and buffer sizing. In *Formal Methods and Models for Codesign (MEMOCODE), International Conference on*. IEEE, 2014.

[53] Thomas A Henzinger, Xavier Nicollin, Joseph Sifakis, and Sergio Yovine. Symbolic model checking for real-time systems. *Information and computation*, 111(2):193–244, 1994.

[54] Thomas A Henzinger and Joseph Sifakis. The embedded systems design challenge. In *International Symposium on Formal Methods*, pages 1–15. Springer, 2006.

[55] Michael Hübner and Jürgen Becker. *Multiprocessor system-on-chip: hardware design and tool integration.* Springer Science & Business Media, 2010.

[56] Aya Ibrahim, Alena Simalatsar, Stefanos Skalistis, Federico Angiolini, Marcel Arditi, Jean-Philippe Thiran, and Giovanni De Micheli. Assessment of image quality vs. computation cost for different parameterizations of ultrasound imaging pipelines. In *Proceedings of the 6th Workshop on Medical Cyber-Physical Systems*, number EPFL-CONF-209034, 2015.

[57] Ahmed Jerraya and Wayne Wolf. *Multiprocessor systems-on-chips*. Elsevier, 2004.

[58] Daniel Johnson, Matthew Johnson, John Kelm, William Tuohy, Steven Lumetta, and Sanjay Patel. Rigel: A 1,024-core single-chip accelerator architecture. *IEEE Micro*, 31(4):30–41, 2011.

[59] I. Kadayif, M. Kandemir, and U. Sezer. An integer linear programming based approach for parallelizing applications in on-chip multiprocessors. DAC '02, pages 703–706, New York, NY, USA, 2002. ACM.

[60] Kalray. Kalray MPPA-256, 2015. Available at http://www.kalray.eu/IMG/pdf/FLYER_MPPA_MANYCORE.pdf.

[61] RE Kessler. The cavium 32 core octeon ii 68xx. In *Hot Chips 23 Symposium (HCS), 2011 IEEE*, pages 1–33. IEEE, 2011.

[62] Nam Sung Kim, Todd Austin, David Baauw, Trevor Mudge, Krisztián Flautner, Jie S Hu, Mary Jane Irwin, Mahmut Kandemir, and Vijaykrishnan Narayanan. Leakage current: Moore's law meets static power. *computer*, 36(12):68–75, 2003.

[63] Namhoon Kim, Bryan C Ward, Micaiah Chisholm, Cheng-Yang Fu, James H Anderson, and F Donelson Smith. Attacking the one-out-of-m multicore problem by combining hardware management with mixed-criticality provisioning. In *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2016 IEEE*, pages 1–12. IEEE, 2016.

[64] Angeliki Kritikakou, Christine Rochange, Madeleine Faugère, Claire Pagetti, Matthieu Roy, Sylvain Girbal, and Daniel Gracia Pérez. Distributed run-time WCET controller for concurrent critical tasks in mixed-critical systems. In *Proceedings of the 22nd International Conference on Real-Time Networks and Systems*, page 139. ACM, 2014.

[65] Tadahiro Kuroda. Cmos design challenges to power wall. In *Microprocesses and Nanotechnology Conference, 2001 International*, pages 6–7. IEEE, 2001.

[66] Jean-Yves Le Boudec and Patrick Thiran. *Network calculus: a theory of deterministic queuing systems for the internet*, volume 2050. Springer Science & Business Media, 2001.

[67] E. Lee. Consistency in dataflow graphs. *Parallel and Distributed Systems, IEEE Transactions on*, 2(2):223–235, Apr 1991.

[68] Edward A Lee and David G Messerschmitt. Synchronous data flow. *Proceedings of the IEEE*, 75(9):1235–1245, 1987.

[69] Julien Legriel. *Multi-Criteria Optimization and its Application to Multi-Processor Embedded Systems*. PhD thesis, université de Provence, 2011.

[70] Julien Legriel, Colas Le Guernic, Scott Cotton, and Oded Maler. Approximating the pareto front of multi-criteria optimization problems. In *TACAS*. Springer, 2010.

[71] Renato Mancuso, Rodolfo Pellizzoni, Neriman Tokcan, and Marco Caccamo. Wcet derivation under single core equivalence with explicit memory budget assignment. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 76. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

[72] Andrew Nelson, Kees Goossens, and Benny Akesson. Dataflow formalisation of real-time streaming applications on a composable and predictable multi-processor soc. *Journal of Systems Architecture*, 61(9):435–448, 2015.

[73] Viet Anh Nguyen, Damien Hardy, and Isabelle Puaut. Scheduling of parallel applications on many-core architectures with caches: bridging the gap between WCET analysis and schedulability analysis. In *9th Junior Researcher Workshop on Real-Time Computing (JRWRTC 2015)*, Lille, France, Nov 2015.

[74] Viet Anh Nguyen, Damien Hardy, and Isabelle Puaut. Cache-conscious offline real-time task scheduling for multi-core processors. In *29th Euromicro Conference on Real-Time Systems, ECRTS 2017, June 27-30, 2017, Dubrovnik, Croatia*, pages 14:1–14:22, 2017.

[75] Xavier Nicollin, Joseph Sifakis, and Sergio Yovine. Compiling real-time specifications into extended automata. *IEEE transactions on Software Engineering*, 18(9):794–804, 1992.

[76] Jan Nowotsch. *Interference-sensitive worst-case execution time analysis for multicore processors*. PhD thesis, 2014.

[77] Jan Nowotsch, Michael Paulitsch, Daniel Bühler, Henrik Theiling, Simon Wegener, and Michael Schmidt. Multi-core interference-sensitive wcet analysis leveraging runtime resource capacity enforcement. In *Real-Time Systems (ECRTS), 2014 26th Euromicro Conference on*, pages 109–118. IEEE, 2014.

[78] Jan Nowotsch, Michael Paulitsch, Arne Henrichsen, Werner Pongratz, and Andreas Schacht. Monitoring and wcet analysis in cots multi-core-soc-based mixed-criticality systems. In *Proceedings of the conference on Design, Automation & Test in Europe*, page 67. European Design and Automation Association, 2014.

[79] Fermi NVidia. Nvidia's next generation cuda compute architecture. *NVidia, Santa Clara, Calif, USA*, 2009. Available at https://www.nvidia.com/content/PDF/fermi_white_papers/NVIDIA_Fermi_Compute_Architecture_Whitepaper.pdf.

[80] Andreas Olofsson. Epiphany-v: a 1024 processor 64-bit risc system-on-chip. *arXiv preprint arXiv:1610.01832*, 2016.

[81] R. Pellizzoni, E. Betti, S. Bak, G. Yao, J. Criswell, M. Caccamo, and R. Kegley. A predictable execution model for cots-based embedded systems. In *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2011 17th IEEE*, pages 269–279, April 2011.

[82] Michael L. Pinedo. *Scheduling: Theory, Algorithms, and Systems.* Springer Publishing Company, Incorporated, 3rd edition, 2008.

[83] Dumitru Potop-Butucaru and Isabelle Puaut. Integrated worst-case execution time estimation of multicore applications. In *OASIcs-OpenAccess Series in Informatics*, volume 30. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2013.

[84] ARGO project. Wcet-aware parallelization of model-based applications for heterogeneous parallel systems.

[85] Isabelle Puaut. Cache analysis vs static cache locking for schedulability analysis in multitasking real-time systems. *Proc. WCET, Vienna, Austria*, 2002.

[86] Peter Puschner and Alan Burns. Guest editorial: A review of worst-case execution-time analysis. *Real-Time Systems*, 18(2):115–128, 2000.

[87] Carl Ramey. Tile-gx100 manycore processor: Acceleration interfaces and architecture. In *Hot Chips 23 Symposium (HCS), 2011 IEEE*, pages 1–21. IEEE, 2011.

[88] Paul Regnier, George Lima, Ernesto Massa, Greg Levin, and Scott Brandt. Run: Optimal multiprocessor real-time scheduling via reduction to uniprocessor. In *Real-Time Systems Symposium (RTSS)*, pages 104–115. IEEE, 2011.

[89] Hamza Rihani, Matthieu Moy, Claire Maiza, Robert I. Davis, and Sebastian Altmeyer. Response time analysis of synchronous data flow programs on a many-core processor. In *Proceedings of the 24th International Conference on Real-Time Networks and Systems*, RTNS '16, pages 67–76, New York, NY, USA, 2016. ACM.

[90] Benjamin Rouxel, Steven Derrien, and Isabelle Puaut. Tightening contention delays while scheduling parallel applications on multi-core architectures. *ACM Trans. Embed. Comput. Syst.*, 16(5s):164:1–164:20, September 2017.

[91] S. Saidi, R. Ernst, S. Uhrig, H. Theiling, and B. D. de Dinechin. The shift to multicores in real-time and safety-critical systems. In *Hardware/Software Codesign and System Synthesis (CODES+ISSS), 2015 International Conference on*, pages 220–229, Oct 2015.

[92] Robert R Schaller. Moore's law: past, present and future. *IEEE spectrum*, 34(6):52–59, 1997.

[93] Lars Schor, Hoeseok Yang, Luliana Bacivarov, and Lothar Thiele. Expandable process networks to efficiently specify and explore task, data, and pipeline parallelism. In *Compilers, Architecture and Synthesis for Embedded Systems (CASES), 2013 International Conference on*, pages 1–10. IEEE, 2013.

[94] Andreas Schranzhofer, Jian-Jia Chen, and Lothar Thiele. Timing predictability on multi-processor systems with shared resources. In *Embedded Systems Week-Workshop on Reconciling Performance with Predictability*, page 87, 2009.

[95] A. Simalatsar, R. Passerone, and D. Densmore. A methodology for architecture exploration and performance analysis using system level design languages and rapid architecture profiling. In *Industrial Embedded Systems, 2008. SIES 2008. International Symposium on*, pages 95–102, June 2008.

[96] Richard L Sites and Anant Agarwal. Multiprocessor cache analysis using atum. *ACM SIGARCH Computer Architecture News*, 16(2):186–195, 1988.

[97] Stefanos Skalistis, Federico Angiolini, Alena Simalatsar, and Giovanni De Micheli. Safe and efficient deployment of data-parallelisable applications on many-core platforms: Theory and practice (minor-revion). *IEEE Design & Test*, 2017.

[98] Stefanos Skalistis and Alena Simalatsar. Modeling of reconfigurable medical ultrasonic applications in bip. In *5th Workshop on Medical Cyber-Physical Systems*, page 66, 2014.

[99] Stefanos Skalistis and Alena Simalatsar. Worst-case execution time analysis for many-core architectures with NoC. In *International Conference on Formal Modeling and Analysis of Timed Systems, FORMATS*, 2016.

[100] Stefanos Skalistis and Alena Simalatsar. Near-optimal deployment of dataflow applications on many-core platforms with real-time guarantees. In *2017 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 752–757. IEEE, 2017.

[101] Anand Srinivasan and Sanjoy Baruah. Deadline-based scheduling of periodic task systems on multiprocessors. *Information Processing Letters*, 84(2):93–98, 2002.

[102] Vivy Suhendra and Tulika Mitra. Exploring locking & partitioning for predictable shared caches on multi-cores. In *Proceedings of the 45th annual Design Automation Conference*, pages 300–303. ACM, 2008.

[103] Lili Tan. The worst case execution time tool challenge 2006: The external test. In *Leveraging Applications of Formal Methods, Verification and Validation, 2006. ISoLA 2006. Second International Symposium on*, pages 241–248. IEEE, 2006.

[104] Pranav Tendulkar. *Mapping and Scheduling on Multi-core Processors using SMT Solvers*. PhD thesis, Universite de Grenoble I-Joseph Fourier, 2014.

[105] Pranav Tendulkar, Peter Poplavko, Ioannis Galanommatis, and Oded Maler. Many-core scheduling of data parallel applications using smt solvers. In *Digital System Design (DSD), 2014 17th Euromicro Conference on*, pages 615–622. IEEE, 2014.

[106] Pranav Tendulkar, Peter Poplavko, and Oded Maler. Symmetry breaking for multi-criteria mapping and scheduling on multicores. In *Formal Modeling and Analysis of Timed Systems*, pages 228–242. Springer, 2013.

[107] Pranav Tendulkar, Peter Poplavko, Jules Maselbas, Ioannis Galanommatis, Oded Maler, et al. A runtime environment for real-time streaming applications on clustered multi-cores. Technical report, Technical report, Verimag, 2015.

[108] Texas Instruments Inc. The 66AK2H12 keystone II Processor.

[109] William Thies and Saman Amarasinghe. An empirical characterization of stream programs and its implications for language and compiler design. PACT '10, pages 365–376, New York, NY, USA, 2010. ACM.

[110] Simon Wegener. Towards multicore wcet analysis. In *OASIcs-OpenAccess Series in Informatics*, volume 57. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

[111] Andreas Weichslgartner, Deepak Gangadharan, Stefan Wildermann, Michael Glaß, and Jürgen Teich. Daarm: Design-time application analysis and run-time mapping for predictable execution in many-core systems. In *Hardware/Software Codesign and System Synthesis (CODES+ ISSS), 2014 International Conference on*, pages 1–10. IEEE, 2014.

[112] Reinhard Wilhelm, Jakob Engblom, Andreas Ermedahl, Niklas Holsti, Stephan Thesing, David Whalley, Guillem Bernat, Christian Ferdinand, Reinhold Heckmann, Tulika Mitra, et al. The worst-case execution-time problem—overview of methods and survey of tools. *ACM Transactions on Embedded Computing Systems (TECS)*, 7(3):36, 2008.

[113] Wayne Wolf, Ahmed Amine Jerraya, and Grant Martin. Multiprocessor system-on-chip (mpsoc) technology. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 27(10):1701–1713, 2008.

[114] G. Yao, G. Buttazzo, and M. Bertogna. Feasibility analysis under fixed priority scheduling with fixed preemption points. In *Embedded and Real-Time Computing Systems and Applications (RTCSA), 2010 IEEE 16th International Conference on*, pages 71–80, Aug 2010.

[115] Jun Zhu, Ingo Sander, and Axel Jantsch. Buffer minimization of real-time streaming applications scheduling on hybrid cpu/fpga architectures. In *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '09, pages 1506–1511, 3001 Leuven, Belgium, Belgium, 2009. European Design and Automation Association.

Curriculum vitae

PERSONAL INFORMATION

Stefanos Skalistis

Av. 24 Janvier 34, 1020 Renens (Switzerland)

+41788744658

s.skalistis@gmail.com

http://www.linkedin.com/in/sskalist

Skype sskalist

Sex Male | Date of birth 05/08/1985 | Nationality Greek

PREFERRED JOB

Researcher / Software Engineer

WORK EXPERIENCE

09/2013–Present

PhD Candidate

École Polytechnique Fédérale de Lausanne (EPFL), Lausanne (Switzerland)

Adaptive & Efficient Hard Real-time Systems

Carry out research on the deployment and execution of highly-parallelisable applications with real-time constraints on many-core architectures with network-on-chip (NoC):

- Interference-sensitive Worst-Case completion time and total latency guarantees
- Safe and Near-optimal Deployment of Data-flow Applications
- Safe and Optimal Runtime Optimisations for Data-flow Applications
- Power Management
- Safe Quality-of-Service control / Mixed-criticallity execution

(Funded by the Nano-tera.ch initiative, as part of the UltrasoundToGo project)

10/2012–04/2013

Senior Reseach Associate

Coventry University Enterprises
Priory Street, Coventry, CV1 5FB (United Kingdom)

As part of a project funded by the UK's Ministry of Defense:

- Carry out research on exploiting Health & Usage Monitoring Systems (HUMS) to automate Military Logistics Platform
- Design and implementation of gear-shifting controller for heavy duty vehicles operating under extreme heat conditions, using HUMS data
- Re-design existing semi-autonomous control system and integration of of gear-shifting controller
- Develop an interactive simulation to investigate the impact of Human Factors on Military Convoys

07/2010–07/2012

Research Associate

Aristotle University of Thessaloniki, Thessaloniki (Greece)

- Research on creating/adapting software components from existing source code using Model-Driven techniques
- Lead developer of the "Component Adaptation Environment" tool chain

08/2008–08/2009

Software Developer and System Administrator

Research and Informatics Office of the Hellenic Army (Greece)

Activities are under a 10-year non-disclosure agreement.

Business or sector Public administration and defence; compulsory social security

05/2008–10/2008

Software Developer

Aristotle University of Thessaloniki
Panepistimioupoli, Thessaloniki (Greece)

Design and develop a Software Quality assessment OSGi plugin for the Software Quality Observatory of Open Source Software (SQO-OSS) platform

15/12/2007–30/07/2008

Software Analyst and Developer

Aristotle University of Thessaloniki
Panepistimioupoli, Thessaloniki (Greece)

Manage, analyse and develop the autonomous Navigation and AI planning modules of an autonomous robot for search & rescue operations.

01/09/2006–15/05/2008

Software Analyst and Developer

Hellenic Petroleum SA., Thessaloniki (Greece)

Analyse, design and develop an automated monitoring system of crude oil distribution

Business or sector Oil Industry

EDUCATION AND TRAINING

01/09/2013–Present

PhD Candidate

PhD Degree

École Polytechnique Fédérale de Lausanne, Lausanne (Switzerland)

01/09/2009–11/02/2011

MSc in Informatics (GPA 9.89 / 10.0)

Masters Degree

Aristotle University of Thessaloniki, Thessaloniki (Greece)

01/09/2003–01/02/2008

BSc in Informatics (GPA 9.11 / 10.0)

Bachelor Degree

Aristotle University of Thessaloniki, Thessaloniki (Greece)

PERSONAL SKILLS

Mother tongue(s) Greek

Other language(s)

	UNDERSTANDING		SPEAKING		WRITING
	Listening	Reading	Spoken interaction	Spoken production	
English	C1	C1	C1	B2	C1

Levels: A1 and A2: Basic user - B1 and B2: Independent user - C1 and C2: Proficient user
Common European Framework of Reference for Languages

Communication skills

- Good communication skills (gained through perpetual communication with team members and projects' partners)
- Fast learner (gained through the diversity of IT fields)
- Team spirit

Organisational / managerial skills

- Leadership (Lead developer on the "OPEN-SME" EU funded project, Chair of A.U.Th. ACM

Student Chapter, President of AEGEL association)

- Good experience in project or team management (Course in Msc degree, Lead developer on "OPEN-SME" EU funded project)
- Good at prioritizing (Research positions & Lead developer on "OPEN-SME" EU funded project)

Job-related skills

- Major experience in Scheduling and Real-time Systems (as part of PhD Thesis)
- Major experience in Software Reuse Processes & Tools (as part of work of "OPEN-SME" EU funded Project)
- Major experience in Domain Engineering (Master Thesis)
- Major experience in Source Code Analysis, Evaluation and Generation (as part of work on two EU funded Projects)
- Experience in Open Source Software (as part of work on two EU funded Projects)
- Experience in writing Deliverables (as part of work of "OPEN-SME" EU funded Project)
- Minor experience in Proposal Writing
- Experience in Robotics (as a member of the Pandora Team)
- Experience in Industry & Automation (as part of the BSc Thesis)
- Familiar with most of the classic methods of Software Development, gained through years of interaction with the Software Engineering Groups (BSc & MSc Thesis, EU funded projects, etc)
- Experience in Extreme Programming & Test-first Design (Bachelor Thesis)

Digital competence

- Excellent in Component-Based Software Engineering
- Good use of various Technologies & Tools (SOA, OSGi, ORM, RPC, Bytecode engineering)
- Excellent use of Eclipse & Visual Studio IDE
- Excellent use of Object Oriented Programming Languages (Java, C++/C#, VB.NET)
- Excellent in meta-programming
- Good use of scripting languages (Python, PHP JavaScript, etc)
- Good use of databases (Oracle, MySQL, MSSQL, PostGres)
- Excellent use of OS (Windows, Linux)
- Excellent use of Office Application (Microsoft, LibreOffice)

Other skills

- Paint-Ball & Fencing
- Chess
- Robotics

ADDITIONAL INFORMATION

Publications

- **Skalistis, S., Angiolini, F., Simalatsar, A. & De Micheli, G. "Safe and Efficient Deployment of Data-Parallelisable Applications on Many-Core Platforms: Theory and Practice". In IEEE Design & Test- Special issue on Time-Critical System Design**
- **Skalistis, S. & Simalatsar, A. (2017, Mar). "Near-optimal Deployment of Dataflow Applications on Many-core Platforms with Real-time Guarantees". In *Design, Automation and Test in Europe Conference and Exhibition (DATE), 2017,* (pp. 752-757). IEEE.**
- **Skalistis, S. & Simalatsar, A. (2016, Aug). "Worst-Case Execution Time Analysis for Many-Core Architectures with NoC." In *14th International Conference Formal Modeling and Analysis of Timed Systems***
- **Ibrahim, A., Simalatsar, A., Skalistis, S., Angiolini, F., Arditi, M., Thiran, J. P., & De Micheli, G. (2015, Apr). "Assessment of Image Quality vs. Computation Cost for Different Parameterizations of Ultrasound Imaging Pipelines." In *6th Workshop on Medical Cyber-Physical Systems***
- **Skalistis, S. & Simalatsar, A. (2014, Apr). "Modeling of Reconfigurable Medical Ultrasonic**

Applications in BIP." In *5th Workshop on Medical Cyber-Physical Systems* (p. 66).

- Skalistis, S., Petrovic, D., & Shaikh, S. A. (2013, Oct). "Operating heavy duty vehicles under extreme heat conditions: A fuzzy approach for smart gear-shifting strategy." *16th International IEEE Conference onIntelligent Transportation Systems (ITSC), 2013* (pp. 961-966). IEEE.
- Gempton, N., Skalistis, S., Furness, J., Shaikh, S., & Petrovic, D. (2013). "Autonomous control in military logistics vehicles: Trust and safety analysis." In *Engineering psychology and cognitive ergonomics. Applications and services* (pp. 253-262). Springer
- Kakarontzas, G., Stamelos, I., Skalistis, S., & Naskos, A. (2012, Sep). "Extracting Components from Open Source: The Component Adaptation Environment (COPE) Approach." *38th EUROMICRO Conference onSoftware Engineering and Advanced Applications (SEAA),* (pp. 192-199). IEEE.

Projects

EU funded projects:

- Open Source Reuse Services for SMEs (http://opensme.eu/)
- Software Quality Observatory for Open Source Software (http://www.sqo-oss.org/)

Nano-tera.ch funded projects:

- UltrasoundToGo (http://www.nano-tera.ch/projects/359.php)

Honours and awards

- Award as top graduate student of 2010 (Information Systems) class (2011)
- Received scholarship for the Masters Programme in Informatics, Emphasis in Information Systems (2010)
- Excellency and Innovation award from the Aristotle University of Thessaloniki (2009)
- Award and Scholarship as top undergraduate student of 2003 class (2008)