Constraint programming for design space exploration of dataflow applications on multi-bus architectures

Amna Gharbi

INSTITUT POLYTECHNIQUE DE PARIS

NNT : 2021IPPAT018

Thèse de doctorat

Constraint Programming for Design Space Exploration of Dataflow Applications on Multi-Bus Architectures

Thèse de doctorat de l'Institut Polytechnique de Paris
préparée à Télécom Paris

École doctorale n°626 Institut Polytechnique de Paris (IPP)
Spécialité de doctorat : Informatique

Thèse présentée et soutenue à Sophia Antipolis, le 10 Novembre, 2021, par

Amna Gharbi

Composition du Jury :

Alix Munier-Kordon Professeure, Sorbonne Université	Présidente
Sebastien Voss Professeur, FH Aachen, University of Applied Sciences	Rapporteur
Alain Girault Directeur de recherche, INRIA Grenoble	Rapporteur
Frédéric Mallet Professeur, Université Côte d'Azur	Examinateur
Ludovic Apvrille Professeur, Télécom Paris	Directeur de thèse
Renaud Pacalet Directeur d'étude, Télécom Paris	Co-directeur de thèse
Andrea Enrici Ingénieur de recherche, Nokia Bell Labs France	Co-encadrant de thèse

Remerciements

« L'interdépendance est une valeur qui surpasse l'indépendance. »

Stephen R. Covey

Ce travail est le résultat de mes efforts mais c'est sans doute grâce au soutien et à l'accompagnement dont j'ai pu bénéficier que ce travail a vu le jour. C'est pour cela que je dédie les premiers mots de ce manuscrit à toutes ces personnes qui ont contribué à son aboutissement.

J'exprime toute ma reconnaissance à mes directeurs de thèse, Ludovic Apvrille et Renaud Pacalet, pour m'avoir accueilli au sein du labSoC. Ils m'ont fourni tout le soutien nécessaire pour réussir mon travail de recherche. Leurs conseils et leurs remarques précieuses sont pour beaucoup dans le résultat final de ce travail. Enfin, leurs nombreuses relectures et corrections de cette thèse ont été très appréciables.

Je remercie tout particulièrement Andrea Enrici de Nokia Bell Labs qui a toujours été disponible et s'est toujours intéressé à l'avancée de mes travaux. Ses encouragements m'ont aidé à persévérer dans les moments difficiles et ont gratifié les efforts que j'ai fournis tout au long de ma recherche.

Ensuite, je tiens à remercier tous les membres de mon jury pour le temps qu'ils ont consacré pour évaluer mon travail, ainsi que pour toutes les connaissances et l'expertise qu'ils ont apportées. Je remercie mes rapporteurs: Alain Girault et Sebastien Voss d'avoir pris le temps de lire ma thèse et de donner des remarques qui ont contribué à son amélioration, et les examinateurs : Alix Munier-Kordon et Frédéric Mallet pour l'intérêt qu'ils portent à ma recherche.

Un remerciement tout particulier va aux personnes qui ont égayé mes journées au labSoC. Merci à : Maysam, Matteo, Benjamin, Letitia, Minh et Le Van pour tous les moments que nous avons partagés. J'ai eu beaucoup de chance d'être entourée par vous pendant ces années. Merci à Rabéa pour toutes les informations utiles qu'elle n'a pas hésité à partager avec moi et les précieux conseils qu'elle m'a donnés. Merci à Tullio pour les discussions enrichissantes pendant les pauses-déjeuner.

Ce travail n'aurait certainement pas vu le jour sans le soutien et les sacrifices immenses que mes parents, Najib et Malika, ont fourni. Leurs encouragements se sont inscrits dans ma tête et ont résonné à chaque épreuve que j'ai pu traverser me donnant force et détermination pour atteindre mes objectifs. Ils m'ont toujours poussé à apprendre et étaient le flambeau qui illuminait mon chemin. Merci maman et papa. Merci à mon frère, Ahmed, de m'avoir toujours diffusé un rayon de joie et de bonne humeur.

Enfin, merci à mon partenaire et époux, Mohamed, pour son soutien inconditionnel et son enthousiasme contagieux. Merci de m'avoir donné tout le soutien et l'amour pour donner le meilleur de moi-même. Merci d'avoir été ma joie et ma source de motivation quand elles me manquaient.

Résumé substantiel

Le travail de cette thèse s'inscrit dans le cadre d'une collaboration entre Télécom Paris et Nokia Bell Labs France. Dans ce contexte, nous nous intéressons aux systèmes de traitement du signal. Un système de traitement du signal peut être perçu comme un système qui reçoit un signal d'entrée, par exemple, une vidéo ou des images, y applique un certain traitement, et produit un signal de sortie. Dans les systèmes modernes de traitement du signal, plusieurs unités de calcul hétérogènes sont embarquées sur la même puce et exécutent les tâches des applications en parallèle et d'une manière distribuée. Dans ce contexte, l'infrastructure de communication devient un élément clé pour la distribution des tâches sur les différentes unités de calcul en assurant le transfert des données d'une unité à l'autre.

Pour exploiter pleinement les capacités potentielles d'une architecture multiprocesseur, les décisions sur comment allouer et ordonnancer les tâches sur les unités de calcul, ainsi que les transferts de données sur les ressources de communication doivent être prises en compte. Il existe plusieurs architectures de communication offrant différents avantages et limitations rendant chacune plus ou moins adaptée selon les différents cas d'utilisation. Les architectures de communication couramment utilisées incluent typiquement des bus partagés, ou des crossbars, ou ont recours à du multi-bus avec des topologies personnalisables telles que les architectures hiérarchiques, les topologies en anneau, ou encore les réseaux sur puce.

Bien qu'il existe de nombreux travaux visant à automatiser la synthèse d'une topologie optimale pour les architectures multi-bus, les travaux visant à identifier une allocation et un ordonnancement des tâches et des communications se concentrent majoritairement sur le bus partagé ou sur les réseaux sur puce. Dans cette thèse, nous proposons trois contributions pour combler cette lacune : 1) Une formulation « Satisfiability Modulo Theories » (SMT) qui permet d'explorer les décisions d'allocation et d'ordonnancement sur les architectures multi-bus pour l'optimisation de la latence ; Nous démontrons son applicabilité pour produire des solutions pour des applications connues. 2) Pour améliorer le passage à l'échelle de la recherche optimale de la première contribution, nous proposons une nouvelle technique pour élaguer l'espace des solutions devant être explorées. Notre évaluation démontre un gain sur le passage à l'échelle. Finalement, 3) la consommation d'énergie par

les communications sur bus est étudiée ; nous montrons comment optimiser la latence et la consommation conjointement. Nos évaluations montrent comment différents compromis entre latence et consommation d'énergie peuvent être étudiés. De plus, nous montrons comment nos contributions ont été intégrées à un outil de modélisation et de vérification particulièrement adapté à la conception des systèmes embarqués au niveau système (TTool).

Dans cette thèse, nous considérons les applications de flux de données représentées par des graphes de dépendances, où les nœuds représentent les tâches et les arcs représentent les communications (ou des dépendances de données). Pour les architectures, nous considérons des architectures avec des unités de calcul hétérogènes, par exemple, un CPU, un DSP ou encore un accélérateur matériel spécialisé en un type de traitement particulier. Les unités de calcul peuvent avoir un ou plusieurs cœurs d'exécution. Pour l'architecture de communication, nous considérons les architectures à base d'un ou plusieurs bus. Ces derniers peuvent être disposés selon diverses topologies, par exemple, linéairement, en arbre, en anneau, ou encore d'une manière ad-hoc.

Nous proposons dans la première contribution de cette thèse un modèle de contraintes qui permet d'identifier : 1) une allocation des tâches sur les unités de calcul, 2) une allocation des communications aux routes de bus et éventuellement de bridges, 3) un ordonnancement des tâches sur les unités de calcul, et 4) une allocation des slots temporels sur les bus aux communications concurrentes. Le modèle proposé peut être configuré pour prendre en compte des contraintes de deadline sur les applications étudiées, ou encore pour minimiser la latence.

Pour évaluer la première contribution, nous avons sélectionné des applications de traitement de signal utilisées pour évaluer des travaux connexes, au même niveau d'abstraction. Ces applications sont : un filtre Sobel, un filtre SUSAN, une application RASTA-PLP, et un encodeur JPEG. Nous montrons que l'approche proposée permet d'analyser les décisions d'allocation et d'ordonnancement sur différentes topologies de bus pour l'optimisation de la latence. Nous montrons aussi qu'elle permet d'analyser les contentions sur les bus dues à la présence de plusieurs communications concurrentes. Cependant, les temps de calculs observés montrent une limitation majeure de l'approche vis à vis son passage à l'échelle.

Pour améliorer le passage à l'échelle de la recherche optimale de la première

contribution, nous proposons une nouvelle technique pour élaguer l'espace des solutions. Cette technique consiste à définir des limites temporelles pour les tâches et les communications. Ces limites sont pour une tâche : 1) le temps de début au plus tôt, 2) le temps de fin au plus tôt, 3) le temps de début au plus tard, et 4) le temps de fin au plus tard. Ainsi, lors de l'exploration de l'espace des solutions, uniquement les ordonnancements qui respectent ces limites temporelles sont étudiés.

Nous comparons les temps de calcul obtenus avec la formulation initiale et ceux que nous obtenons après l'application de la technique de réduction. Nous montrons que nous obtenons une réduction significative des temps d'exécution d'au moins un ordre de grandeur.

Dans les deux premières contributions, nous nous sommes uniquement concentrés sur l'optimisation de la latence. Cependant, dans les systèmes du monde réel, il y a souvent plus d'un critère à satisfaire ou à optimiser, souvent en conflit l'un avec l'autre. Ainsi, dans la troisième contribution, nous étudions l'extension de notre approche pour tenir compte d'un deuxième critère. Comme dans cette thèse, nous nous intéressons aux architectures de communication multi-bus, nous considérons la consommation d'énergie par les communications comme deuxième critère. Nous proposons donc trois approches pour étudier conjointement l'optimisation de la latence et de la consommation.

La première approche peut être utilisée quand on recherche une solution qui minimise la latence en priorité et qui permet des réductions de la consommation mais pas au détriment de la latence. Pour cela, nous nous appuyons sur la méthode d'ordre lexicographique qui consiste à effectuer une première phase d'optimisation de latence, puis une deuxième phase de minimisation de la puissance sous la contrainte de latence minimale. Bien sûr, la deuxième phase n'a de sens que lorsqu'il existe plusieurs solutions minimisant la latence.

La deuxième approche permet d'obtenir des solutions minimisant la consommation tout en respectant des deadlines définis a priori. Pour ce faire, nous nous appuyons sur la méthode de la fonction objectif bornée qui minimise l'énergie sous une contrainte de latence.

Finalement, la troisième approche vise à produire un ensemble de solutions optimales de Pareto en se basant sur une variante de la méthode dite « epsilon-constraint ». Dans cette approche, la solution la plus appropriée peut être sélection-

née parmi l'ensemble des solutions de Pareto généré.

Nous montrons que les trois approches permettent de générer différents compromis entre la latence et la consommation. Le passage à l'échelle de l'optimisation bi-critère représente la limitation majeure de la troisième contribution, surtout en ce qui concerne la troisième approche visant à générer un ensemble de solutions de Pareto.

Pour conclure, cette thèse s'intéresse aux architectures à base de bus mutiples. Pour ce type de plateformes, les contributions déjà proposées étaient insuffisantes. Le travail accompli dans le cadre de cette thèse comble cette lacune en permettant d'étudier les décisions d'allocation et d'ordonnancement dans ces architectures. La technique de réduction proposée apporte une amélioration significative des temps d'exécution. La troisième contribution démontre la flexibilité de l'approche proposée pour encoder différentes approches d'optimisation bi-critères et générer différentes solutions de Pareto.

Nous identifions deux axes principaux pour les perspectives de ce travail. Le premier axe porte sur l'extension de la formulation actuelle pour modéliser d'autres aspects des systèmes étudiés. Par exemple, la consommation d'énergie des unités de calcul ou encore la présence de mémoire partagée dans l'architecture. Le deuxième axe concerne l'élaboration de nouvelles techniques pour améliorer davantage le passage à l'échelle. Pour cela, nous discutons de quelques idées comme la décomposition temporelle et la rupture de symétrie.

Contents

List of Figures

List of Tables

Glossary

CP Constraint Programming. 39, 40, 43–45, 100

CPU Central Processing Unit. 22, 31, 42, 54, 62, 71, 72, 74, 91

DAG Directed Acyclic Graph. 7, 52, 53, 60, 81, 85

DMA Direct Memory Access. 6, 31, 41, 54–56

DSE Design Space Exploration. 3, 8, 19–21, 35–37, 39, 40, 44, 47–49, 59, 65, 69, 73–75, 79, 80, 91, 92, 94–97, 99

DSP Digital Signal Processor. 13, 22, 62, 71, 72, 74, 91

FFT Fast Fourier Transform. 53

FPGA Field Programmable Gate Array. 13, 22, 28, 31, 42, 47

GPU Graphics Processing Unit. 22

HSDF Homogeneous Synchronous Data-Flow. 40, 42

ILP Integer Linear Programming. 38, 40, 41, 43–45, 100

LP Linear Programming. 38, 39

MILP Mixed-Integer Linear Programming. 38, 42, 45

MPSoC Multi-Processor System on a Chip. 13, 15, 20, 22, 74, 91

NoC Network on a Chip. 14, 22, 47, 80, 81

PE Processing Element. 53–56, 60–63

PU Processing Unit. 54–56, 60, 62

RTL Register Transfer Level. 15, 17

SAT Boolean Satisfiability Problem. 30, 39, 40, 42, 45

SDF Synchronous Data Flow. 36, 37, 40–43

SMT Satisfiability Modulo Theories. 3, 4, 8, 9, 26, 28, 30, 34, 39–41, 43–45, 47, 48, 59, 60, 65, 68, 69, 76, 79–81, 88, 92, 94, 96, 97, 99, 100

SysML Systems Modeling Language. 3, 6, 7, 26, 27, 65, 66, 68, 69

UML Unified Modeling Language. 3, 6, 7, 26, 27, 65–69

Chapter 1

Introduction

1.1 Context

The work of this thesis is part of a collaboration between Télécom Paris and Nokia Bell Labs France. In this context, we focus on the system-level Design Space Exploration (DSE) of embedded systems for the execution of signal processing applications. We focus on multi-bus architectures that are well-suited for the deployment of signal processing applications. In this Section, we will first define the scope and aim of the DSE in this thesis, in subsection 1.1.1. Then, we will define the target applications and architectures respectively in subsections 1.1.2 and 1.1.3.

1.1.1 Design Space Exploration

In the systems we target, a DSE process aims at identifying the mapping and scheduling of both application tasks and communications between these tasks, such that design constraints and objectives are met. *Design constraints* refer to the set of requirements that are inherent to application and architecture definitions, and that mapping and scheduling decisions should respect. Basically, design constraints concern the application constraints (e.g., precedence constraints between tasks) and capacities of the architecture (e.g., memory capacity, connectivity scheme). *Design objectives* refer to performance goals, that are, for a given system metric, the proposed solution should either be below or over a given threshold (*Satisfy*), or optimal (*Min-*

imize/*Maximize*). Examples are minimizing latency, satisfying real-time deadlines, minimizing power consumption or maximizing throughput. DSE analyzes different alternatives of mapping and scheduling, and selects for deployment the decisions which are compliant with design constraints, and satisfy best design objectives. This decision-making is a key step in embedded systems design [41] but is also complex and time-consuming [30, 53]. In fact, the majority of DSE problems (e.g., scheduling problems) for a multiprocessor architecture are NP-hard [25, 59].

One way to deal with the complexity of embedded systems design is to raise the level of abstraction from which the design is perceived to the so-called *system-level* [30]. Keutzer et al. [34] suggest that design decisions must be taken at the highest level of abstraction to allow exploiting all the degrees of freedom that are available. In fact, high-level abstractions allow global design methodologies by omitting precise circuit behavior details [30]. The work of this thesis, as well as the discussions in this manuscript, are located at the system-level of abstraction.

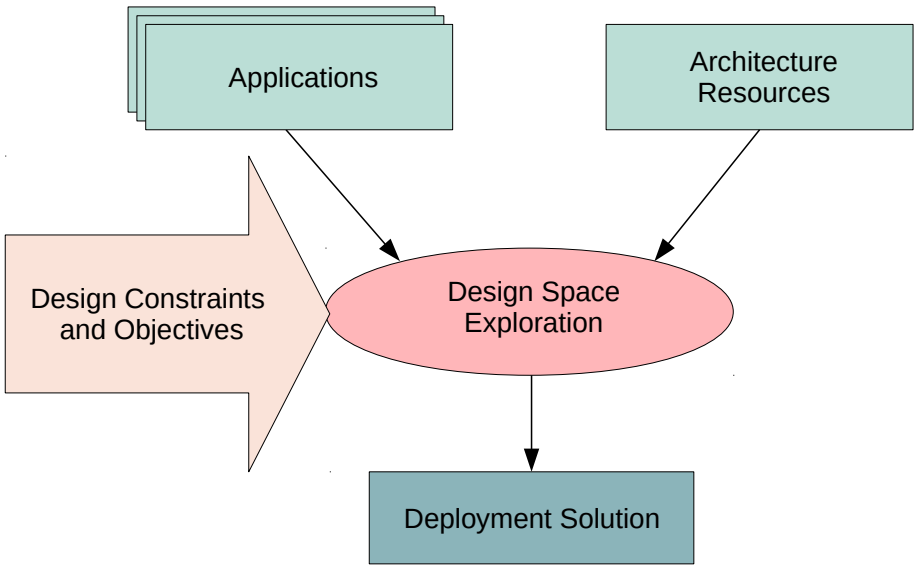


Figure 1.1: The Y-chart approach for DSE [36].

DSE is typically supported by means of automatic or semi-automatic design tools and methods. DSE is often associated with the Y-chart approach [36], illustrated in Figure 1.1. The Y-Chart takes as input one or several applications and the available architecture resources, as well as design constraints and objectives. Subsequently, it relies on analysis and search techniques to navigate the design space and evaluate

potential design candidates. Finally, it outputs a deployment solution that describes the decisions which were taken. In this thesis, a *deployment solution* refers to a *mapping* and a *scheduling* that we define as follows.

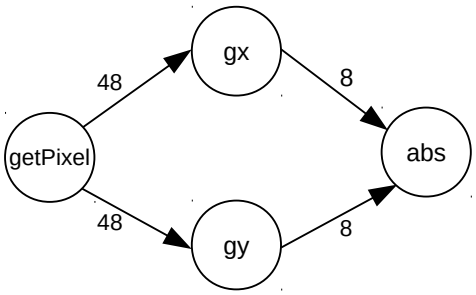
- **Mapping** is the spatial allocation of architecture resources (processing resource, communication resource) and their assignment to application parts (tasks, communications). A mapping concerns the association of tasks to processing resources as well as the mapping of communications to paths (routes) from the architecture.

- **Scheduling** is the temporal allocation of architecture resources to application parts. It refers to the scheduling of tasks on processing resources and the scheduling of data transfers between tasks on communication resources.

1.1.2 Applications

The dataflow paradigm is a very good fit for signal processing applications [21]. A dataflow application consists of a set of processing steps (tasks)—which operate on data to transform it— related by data dependencies (communications), which aim at exchanging data between tasks. Mapping of communications should resolve how data transfers between the related tasks can be performed by architectural components. A dataflow application is specified by means of a dependency graph where nodes represent tasks and edges represent communications between them.

Examples of dataflow applications are *Sobel* and *SUSAN* filtering algorithms illustrated in Figure 1.2. *SUSAN* (Smallest Univalue Segment Assimilating Nucleus), Figure 1.2b, is a noise-reduction algorithm for edge and corner detection in images. It uses nonlinear filtering to reduce noise in an image while preserving its underlying structure [60]. Here, task *direction* produces data which is consumed by task *thin*: after task *direction* has determined the direction of an edge to decide whether or not a pixel is an edge point, a thinning algorithm (*thin*) is applied to the detected edges. *Sobel* is a filtering algorithm used to emphasize edges on an image. It performs a 2-D spatial gradient measurement on an image and emphasizes regions of high spatial frequency which correspond to edges. In a signal processing system, multiple applications, such as *Sobel* and *SUSAN*, can run in parallel and have their tasks and



(a) *Sobel* application model

(b) *SUSAN* application model

Figure 1.2: Example of dataflow application dependency graphs [50]. Nodes represent tasks. Edges represent communications between them, annotated with the amount of data to exchange.

communications compete to shared resources. Besides, applications are often subject to one or several design objectives. In case multiple applications are executed in parallel, different timing design objectives can be associated to each application to account for various levels of criticality. For instance, while the execution of an application A should end before a real-time deadline, the execution of an application B should be as short as possible.

1.1.3 Architectures

A major evolution in embedded system design was enabled by the migration from the single processing paradigm to parallelization [79]. Multi-Processor Systems on a Chip (MPSoCs) were developed to parallelize calculations on multiple processing resources in order to meet design objectives in the areas of communications, signal processing, multimedia, networking [78, 79]. . . To reach the design objectives of applications, MPSoC architectures tend to be heterogeneous such that various types of processing resources (e.g., Digital Signal Processor (DSP), general-purpose Central Processing Unit (CPU), Graphics Processing Unit (GPU)) are integrated within a sin-

gle chip and prompted to work in parallel on various applications calculations (tasks). In this context, communications, in the form of data exchanges between different processing resources, become crucial and represent a key enabler to distribute tasks over multiple processing resources.

There is a range of communication architectures (also referred to as *interconnect* in this thesis) that can be deployed in a MPSoC design, each offering a certain trade-off between design objectives [46], and thus are more or less adapted to the communication needs of a given application. For example, while Networks on a Chip (NoCs) scale better with the number of components connected, shared buses frequently have lower power consumptions and smaller area costs [45]. The shared bus is known as the simplest communication architecture. It consists of parallel wires to which all system components are connected. Only one component can transfer data on the shared bus at any given time. Consequently, increasing the number of components will lead to contentions and performance degradations, which translate to a poor scalability. Crossbar is a communication architecture with a set of buses operating in parallel. It can be partial (i.e., each component is connected to only a subset of the others) or full (i.e., each component is connected to all others) depending on the connectivity requirements. A NoC [7] consists of a network of routers and links where data is sent through the network in form of packets. The multi-bus (also referred to as the *segmented bus* in this manuscript) comes half way between the shared bus and the NoC. It is a bus-based communication architecture where the system bus is split into two or multiple segments (also called buses, or bus segments in this thesis). Each segment connects a subset of system components and operates in parallel to other segments. Different bandwidth capacities can be allocated to each bus segment, depending on the communication needs of components locally connected to the segment. For example, a hierarchical bus can be seen as a segmented bus for which some design restrictions on the structure and protocol have to be respected. In this thesis, we focus on multi-bus communication architectures which will be discussed further in the next Section.

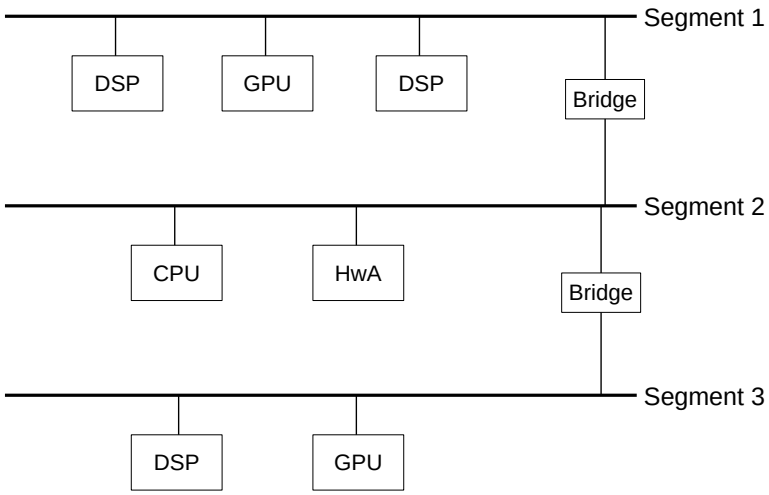


Figure 1.3: An example of a segmented bus communication architecture.

1.1.4 The segmented bus interconnect

The segmented bus is a communication architecture that was first proposed in the late 90's [13] as an alternative to shared bus architectures (i.e., a single bus connecting all system components). These architectures suffer of scalability issues: as the number of connected components increases, the length of the bus wires and the load capacitances increase. In turn, this translates to longer signal propagation delays and to increased power consumption [46].

To overcome this limitation, the segmented bus proposes to split the system bus into segments such that each bus segment behaves as a normal bus, shared between a reduced set of local components. Each segment is allowed to operate in parallel to other segments and unused segments can be deactivated. Segmentation fosters parallel processing and communications, and reduces power consumption, as communications do not need to activate and utilize the entire bus system [81]. Figure 1.3 gives an example of an architecture with a three-bus segments communication architecture. Data can cross as many bus segments as needed to reach the intended destination. For instance, a data transfer between the GPU connected to Segment 1 and the DSP on Segment 3 activates all the bus segments. The inter-segment communications are governed by means of bridge components. A data transfer between the CPU and the hardware accelerator (HwA) connected to

Segment 2 involves only the latter segment to be active. Segment 1 and Segment 3 remain inactive as long as none of their local components is involved in a data transfer.

Multi-bus interconnects are the preferred communication architecture in many designs (e.g., MPEG-4 decoder [73], Spiking Neural Network (SNN) [6]), thanks to improvements they offer in terms of latency, power consumption and area cost. In terms of performance characteristics, they can be positioned halfway between the simple but poorly scalable shared-bus architectures and the more complex NoCs.

In this thesis, we focus on multi-bus (segmented) communication architectures. We aim at providing methods and tools to allow investigating mapping and scheduling of dataflow applications (i.e., signal, image and video processing applications) that allow engineers to evaluate and compare the performance of bus-based interconnects with different topologies (e.g., shared, segmented)

1.2 Problem Statement

Designing a multi-bus architecture, even when the number and type of processing resources have been fixed, comes with a set of entangled issues that need to be solved. These issues include: how many bus segments to create? What is the bandwidth of each bus segment? How to connect segments to each other? Where to place processing resources as well as which tasks to map to these resources? How to map and schedule concurrent communications and avoid contentions? These issues are also typically subject to conflicting design objectives such as minimum latency and power consumption.

Ideally, during design, all these issues would be addressed in an integrated manner so that their inherent interdependencies could be analyzed, and globally optimal design decisions could be taken. However, in practice, a unified solving of all these issues may be prohibitively expensive. Thus, basically, design methodologies focus on solving several issues, separately, like finding an optimized topology and placement of processing resources [17, 55, 56], or address mapping and scheduling but restrict the number of buses to a unique bus [35, 39, 50, 51].

This lack in design methodologies for segmented bus architectures results into an under-utilization of their full potential. For instance, common designs typically

segment buses in only two parts: high-speed and low-speed processing resources, while a larger spectrum of communication schemes could be explored and tailored to the application communication needs. Moreover, lacking systematic methodologies to study application communication needs (e.g., mapping and scheduling concurrent communications) on a target architecture usually leads to oversizing the interconnect bandwidth, that is, wasted cost and area.

In this thesis, we provide methods, techniques and tools for the design space exploration of segmented bus architectures with various topologies. We focus on topologies pre-defined by a user and leave the automation of topology synthesis for future work.

The contributions of this thesis target the following problems:

- How to **efficiently map** and **schedule tasks** and **communications** of **multiple independent applications** on a multi-bus architecture (example in Figure 1.3) such that design objectives on **latency** and **power consumption** are satisfied?
- What is the **minimum bandwidth required** on each bus under latency and power consumption design objectives?

1.3 Contribution

The work of this thesis consists in providing a Constraint Programming (CP)-based DSE approach for tasks and communications of dataflow applications. The proposed approach relies specifically on Satisfiability Modulo Theories (SMT). The aim of the proposed DSE is to find a mapping and scheduling of tasks and communications on an **existing, fixed, multi-bus architecture**. However, it can also be used to guide the **design of new architectures** (e.g., number of buses, interconnect topology), by comparing the found solutions for various architectures and selecting the architecture that best meets the design objectives. In the following, unless otherwise stated, we will use the terms **design** (and **DSE**), for both activities.

Communications between processing resources are allowed to traverse a single bus segment or as many bus segments as needed. If several routes are available between a pair of processing resources, the exploration procedure selects the route that best satisfies the design objectives. The aim of the proposed DSE is to provide

a mapping and scheduling of tasks and communications such that design objectives on latency and power consumption of the multi-bus interconnect are met. A design objective on latency for a given application could be either to satisfy a deadline constraint, or to minimize latency. Multiple applications competing for shared resources can be examined and deployed simultaneously. Besides, we derive estimations on the minimum bandwidth required on each bus segment such that the latency design objective is met and the overall power consumption of buses is minimized. To improve the scalability, we propose a technique which aims at accelerating the joint solving of task and communication scheduling by pruning the temporal domains where variables for the scheduling of tasks and communications are defined. This allows to focus scheduling solutions on temporal domains, which are feasible with respect to precedence relationships and architecture capacities. The proposed technique brings a significant improvement to the DSE run-time.

The steps of the proposed DSE approach are illustrated in Figure 1.4. Steps 2 and 3 denote our contributions. Once applications and architecture models are captured and design objectives are specified in Step 1, Step 2 performs an analysis of the input applications and architecture models to calculate *temporal boundaries* for tasks and communications. This step aims at reducing the run-time spent of Step 4. Step 3 translates input models, design objectives and constraints into a SMT model which describe the joint mapping and scheduling of tasks and communications. This SMT model is optimized thanks to the pre-analysis performed in Step 2. A state-of-the-art SMT solver is used to solve the formulation in Step 4 and return a mapping and scheduling solution if the formulas are satisfiable. If none of the candidates in the design space meets the design constraints and objectives, the user may need to refine the input specification. The output mapping and scheduling solution is optimized with respect to latency and power consumption of the interconnect.

We integrated our approach into a state-of-the-art design environment: TTool [2, 3]. TTool is a free and open-source UML/SysML framework supporting several UML profiles, including DIPLODOCUS [72] that targets hardware/software partitioning. TTool/DIPLODOCUS allows a user to model systems through UML/SysML diagrams and provides automated formal verification, transaction-based simulation and code generation for a given target hardware. However TTool/DIPLODOCUS lacks an engine to perform automatic DSE at system-level. We address this lack by the integra-

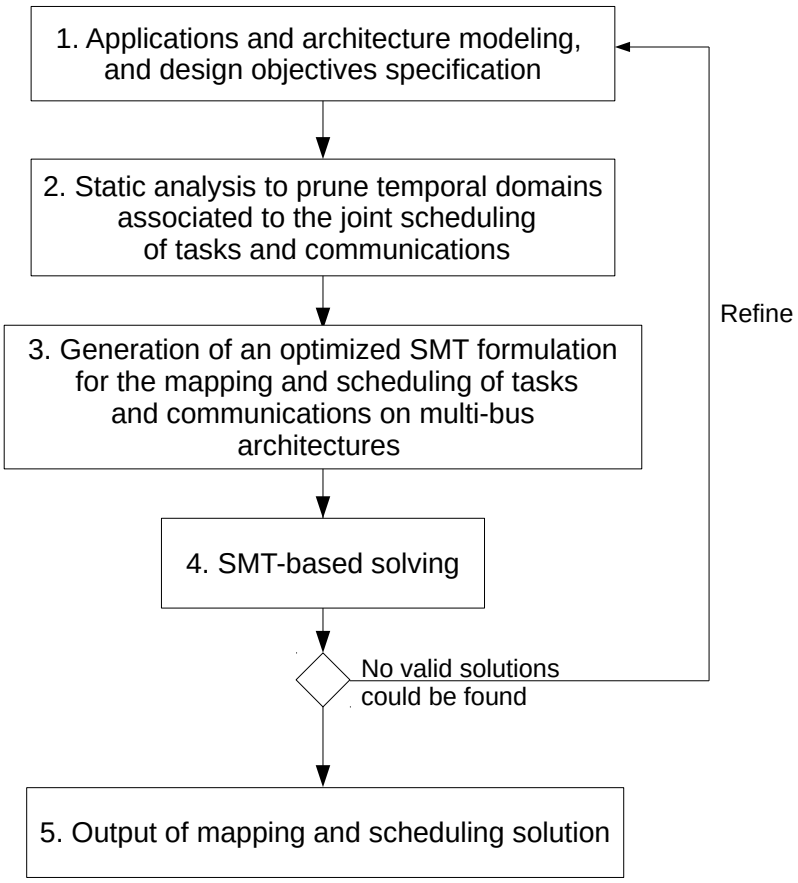


Figure 1.4: An optimized design space exploration approach for the mapping and scheduling of tasks and communications on multi-bus architectures.

tion of our contributions tightly within TTool/DIPLODOCUS.

Figure 1.5 illustrates the software architecture of TTool/DIPLODOCUS after the integration of our DSE. It takes input applications and architecture models with their properties as UML/SysML diagrams, performs system-level DSE according to the flow described in Figure 1.4, and outputs the deployment solution. The output solution can itself represent an input to further investigation at lower abstraction levels like transaction-based simulation. In Figure 1.5, a transformation to C++ code allows to perform simulations for the generated deployment solution.

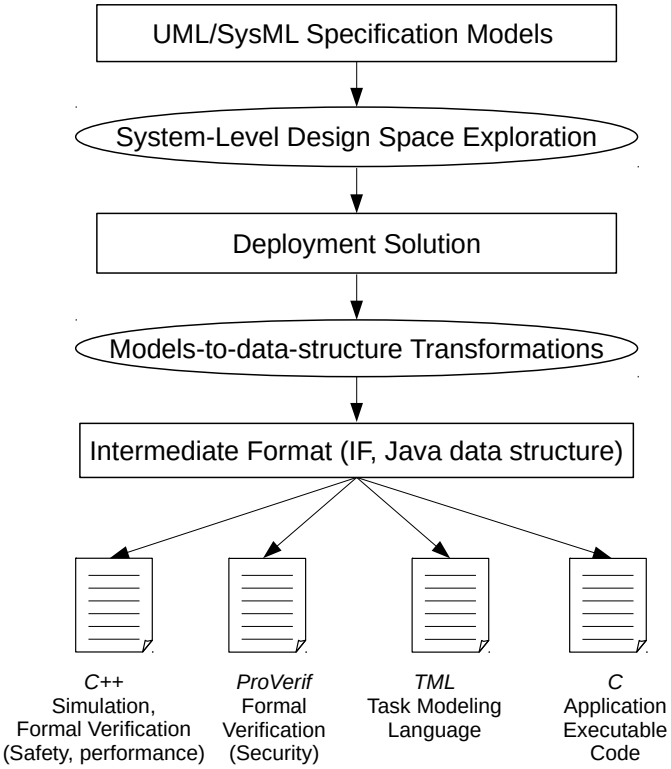


Figure 1.5: The software architecture of TTool for the UML/SysML profile DIPLODOCUS.

1.4 Thesis Outline

The remainder of this thesis is organized as follows.

- In Chapter 2, we identify the components of a DSE approach at system-level and discuss the criteria that define and differentiate DSE problems and solving techniques. Then, we position our contribution according to these criteria and argue our choices. Finally, we review existing DSE works most closely related to ours and highlight the differences with respect to our work.

- Chapter 3 presents our SMT formulation for the DSE of architectures with a multi-bus interconnect. The proposed formulation captures joint tasks and communications mapping and scheduling and allows to analyze their interdependencies. We solve this formulation using a state-of-the-art SMT solver in order to propose a solution which satisfies deadlines or minimize latencies for

multiple applications. The Chapter also presents a case study that shows how the approach efficiently assists the designer in selecting the best interconnect topology for a given set of applications.

- In Chapter 4, our technique to prune the design space and accelerate the SMT solving step is introduced. The speed-up is evaluated for a benchmark of a set of real-world applications. Experiments demonstrate the efficiency of our technique to drastically reduce the DSE run-time.
- Chapter 5 presents our modeling of power consumption of the multi-bus interconnect which has two objectives: first assess the power consumed by buses in order to minimize it. Second, capture the bandwidth requirements of each bus in order to compute the minimum per-bus bandwidth required to achieve the design objectives on latency and power consumption.
- Chapter 6 concludes the manuscript and gives insights into future directions for the work accomplished in this thesis.

Chapter 2

Context and Related Work

In this thesis, we study the Design Space Exploration (DSE) of dataflow applications on architectures with a multi-bus interconnect, at Electronic System-Level (ESL) of abstraction. ESL is a design methodlogy that focuses on high abstraction level. It is defined in [5] as *"the utilization of appropriate abstractions in order to increase comprehension about a system, and to enhance the probability of a successful implementation of functionality in a cost-effective manner, while meeting necessary constraints"*. In this chapter, we focus only on works at ESL of abstraction. The DSE problem consists in the decision problem of finding a mapping and a scheduling of tasks and communications with respect to some design objectives. To solve these problems, DSE techniques are needed. In this Chapter, we have identified two main classes of criteria to organize the related work. The first class concerns aspects that define the DSE problem to be solved, like the target architecture characteristics—and how these are accounted for during mapping and scheduling—or the design objectives to achieve (e.g., minimize latency, minimize power consumption). The second class refers to the techniques proposed to solve the DSE problem like the search strategies used to navigate the design space towards sought solutions. We will discuss these criteria in details in this chapter. Afterwards, we will position our work with respect to the targeted systems. Following that, we will discuss existing work and highlight why it is not suited to solve the problem we study in this thesis. Finally, we conclude and motivate the choices of our work.

The remainder of this Chapter is articulated in five Sections: Sections 2.1 and 2.2

define the criteria that we identified to classify DSE work: Section 2.1 discusses the different aspects and assumptions that define a DSE problem such as the design decisions, the target architecture model and the scheduling solution model. Section 2.2 explains the key components of a DSE approach (e.g., the search strategy) and different alternatives for each component. Section 2.3 positions and motivates our contribution with regard to the criteria discussed in Sections 2.1 and 2.2. In Section 2.4, we review existing DSE work most closely related to our work. Finally, Section 2.5 concludes this chapter and gives an overview of the contributions of this thesis, their motivations, and how our work is different from the existing literature.

2.1 Design Space Exploration Problems

2.1.1 Application model

In this Section, we discuss the Model of Computation (MoC) used to capture applications. A MoC describes the set of rules that govern the execution of an application, by defining the semantics of its components and how they interact. Here, we discuss the Synchronous Data Flow (SDF) paradigm, and other graphs that can be derived by applying transformation to an original SDF graph. We focus on the SDF for its relevance for signal processing applications and for reasons we will discuss in this paragraph.

Synchronous Data Flow

SDF [37] model is very commonly used in DSE work for signal processing applications. SDF is a dataflow paradigm proposed to describe signal processing applications, in a way that concurrency inherent to an application is exposed. The definition of a SDF graph is given in Definition 2.1.1.

Definition 2.1.1 (SDF graph)**.** A SDF graph G is a tuple $\langle A, E \rangle$ where A is a finite set of vertexes denoting computations (called *actors*) and E is a finite set of directed edges denoting First In First Out (FIFO) channels between these actors. Any actor can fire (perform its computation) as soon as enough input data (called token) are

available on its incoming arcs. Edges are annotated with a the number of tokens consumed and produced by each actor firing.

In a SDF graph, production and consumption rates of actors are known a priori. SDF allow for static scheduling at design time, thereby, minimizing run time overhead. In SDF, all computations and data communications can be scheduled statically. During DSE, SDF graphs can be analyzed, as such, or after being transformed to other forms.

Acyclic and Homogeneous Synchronous Data Flow In order to expose the parallelism in an application (i.e., parallelism of computation, and parallelism of data), a special case of SDF, called Homogeneous Synchronous Data-Flow (HSDF), can be obtained by applying a transformation on the original SDF. In a HSDF graph, the amount of tokens exchanged on edges are homogeneous, i.e., production = consumption. HSDF graphs exhibit the potential parallelism in an application. As such, more options for the mapping of actor firing (task) and data exchange (communication) can be explored, compared to an original SDF graph.

Acyclic Homogeneous Synchronous Data-Flow (AHSDF) is a sub-class of SDF where cycles are banned, i.e., there is no path that starts at a given node, traverses a directed sequence of edges and loops back to the same node. By banning cycles, acyclic graphs are useful to isolate a single iteration of the application. Therefore, the AHSDF both exposes the potential parallelism in an application, and allows to isolate a single iteration of the application.

2.1.2 Processing models: heterogeneity

A processing model refers to the processing resources deployed on the architecture. Heterogeneity of a processing model is described with respect to how different deployed processing resources are from each other. We identify three processing models with respect to their level of heterogeneity.

- **Homogeneous:** Processing resources are all identical (e.g., Kalray MPPA-256 platform [69]).

- **Weakly heterogeneous:** Processing resources are all identical, but the clock frequency, memory size or other parameters are different.
- **Strongly heterogeneous:** Processing resources can be of different types (e.g., generic CPU, DSPs, computation-specific hardware accelerators): A processing resource may not be able to support the execution of all applications tasks. A processing resource can contain a single Processing Element (PE) (core) for execution or several cores (e.g., multi-core CPU). Examples of real-word architectures with this processing model are the Samsung Exynos and Qualcomm Snapdragon ranges used in smartphone and tablet architectures.

2.1.3 Communication models

Communications of an application need to be mapped and scheduled on the target communication architecture. A communication model describes how communications between different architecture components are captured during DSE. In existing DSE works, several communication models are considered, ranging from assuming instantaneous communications (i.e., communication overhead is assumed negligible), to deeply analyzing a communication structure in order to estimate communication times, e.g., assignment of bus time slots to communications. In the following bullet points, we first overview communication models, where a communication architecture is analyzed in order to derive a mapping and scheduling of communications.

- **Single shared bus:** This model refers to a single shared bus connecting all components. Here a unique mapping option is available: the shared bus. Since only one component of the bus can have control at a given time, and accounting for the potential parallelism in applications, contention on the shared bus interconnect should be handled during scheduling. Basically, an arbitration policy (e.g., Time Division Multiple Access (TDMA) in [50]) is modeled to estimate communication times on the shared bus.
- **Full/Partial crossbar:** This model corresponds to the crossbar (or bus matrix) interconnect which operates as multiple buses connecting all components (full crossbar), or a subset of components (partial crossbar). Mapping and scheduling communications on a crossbar require to address some constraints such

as exploiting the parallelism of buses while respecting the capacity of communications between pairs of components.

- **Multi-bus - Segmented bus:** This communication model refers to the multi-bus interconnect where multiple bus segments are connected to each other, in a certain scheme (e.g., hierarchical, ring, ad-hoc), and components are connected locally to each segment. It allows parallel transfers on different segments leading to less contentions than the single shared bus. However, mapping and scheduling are more complex because each segment can operate at a different bandwidth, tasks and components can be distributed to bus segments in many different ways, many routes can be available, and inter-segment communications have to be handled along with local communications.

- **NoC**: This model refers to the NoC which is a network structure composed of switches (routers) and links and relying on data packetization: Before being transmitted through the interconnect, data is split into packets. A packet is received by the router in an input buffer, then it is routed to the appropriate output link. Data packets traverse progressively switches and links until reaching the final destination. NoCs are known for being more scalable than bus-based architectures but they are also more complex to design [16]. Therefore, the NoC communication model has been extensively studied [4,52]. DSE for NoCs addresses NoC problems like packetization and depacketization of data.

- **Hybrid:** This category groups works that consider any hybrid combination of buses (shared and segmented), crossbars, and/or NoCs. Mapping and scheduling here account for a meld of design issues from each communication model used.

During DSE, some abstractions can be considered on the analysis of communication mapping and scheduling, resulting in simplified communication models, which do not analyze communications through the interconnect. These models, summarized hereafter, can be applied for any type of communication architecture (e.g., shared bus, NoC).

- **Communication-oblivious:** This model simply assumes that communications between components are instantaneous. Here, the impact of communications on the overall performance is assumed negligible.

- **Fully connected - Constant latency:** This model suggests that all components communicate between each other through dedicated private channels. The communication latency between each pair of components is constant (e.g., defined in a matrix) and is not affected by the co-existing load.

2.1.4 Design decisions

An application is built upon a set of tasks and communications between tasks. An architecture is built upon processing, communication and storage resources. In this thesis, we focus on tasks and communications mapping and scheduling on processing and communication resources. We refer to system parameters that are explored and decided during DSE by *design decisions*. A DSE approach can examine one or several design decisions simultaneously. Basically, the complexity of a DSE problem increases with the number of decisions evaluated at once. However, regarding design decisions in an integrated way allows a holistic view of the system parameters and their interdependencies. This holistic view is often necessary to ensure finding optimal design solutions. In the following, we overview design decisions related to the mapping (application-to-architecture) and the scheduling (application-to-time/order).

- **Task Mapping** refers to the assignment of a processing resource from the architecture to each task from the application.

- **Task Scheduling** refers to determining a temporal organization of tasks. Basically, there are two ways to do this: 1) Determining an order of execution of tasks, or 2) Determining the exact start times of each task. From the latter, the former can be deduced.

- **Communication mapping**, or routing refers to the assignment of one or multiple communication resources from the architecture to each data transfer needed between a pair of tasks from the application.

- **Communication scheduling** refers to determining a temporal organization of communications. As for task scheduling, two ways are possible: 1) Determining an order of execution of communications, or 2) Determining the exact times where communications occur on resources. The former can be deduced from the latter as well.

2.1.5 Design objectives

A design objective is a goal on a cost or a performance metric (criteria) that a design solution should achieve. A design objective can be of two types: 1) *Satisfy*, meaning the metric should be below or over a given threshold, or 2) *Minimize/Maximize*, meaning the optimal value should be reached for the metric. As an example, a real-time system is typically not required to respond as fast as possible (e.g., *Minimize* $latency$) but fast enough so that the timing requirement is respected (e.g., $latency \leq threshold_{latency}$). However, one can be interested in maximizing the usage of the hardware material deployed (e.g., *Maximize* $usage$). Higher exploration effort is required to achieve a *Minimize/Maximize* design objective, as shown for example in [26] for latency, and later in our experimental evaluations (Section 4.4, Chapter 4).

In the rest of this subsection, we overview a non-exhaustive list of the most recurring design objectives that we identified in related work. These design objectives can be considered separately or simultaneously in a DSE problem.

Latency

Performance design objective on latency has received significant research attention, especially in the real-time community. For a dataflow application graph (e.g., Directed Acyclic Graph (DAG)), latency (or end-to-end latency) refers to the time that separates the end of the last task from the start of the first task. Typically, two types of design objectives are applicable to latency: 1) Satisfy, and 2) Minimize. While the latter aims at providing the design solution which guarantees the lowest latency, the former is content with respecting a certain threshold on latency. For example, while some systems have to respond as fast as possible (e.g., in systems where processing resources can only execute a task at once, without preemption, minimizing la-

tency allows to reduce the time during which processing resources are unavailable), other systems only need to respect a certain time budget or a deadline constraint.

Throughput

For periodic applications, throughput is defined as the inverse of the period, i.e., the time it takes to execute an entire iteration of the SDF application graph. Achieving a certain goal for throughput is also a major design objective. As periodic dataflow applications (e.g., streaming applications) run continuously, it is important to assess their data processing rate. Similarly to latency, throughput can be either subject to *Satisfy* constraints (e.g., guaranteeing a minimum throughput), or to *Maximize* objective.

Power consumption

Power consumption is another important design concern [44]. Low power consumption is a key enabler for portable embedded devices (e.g., smartphones), where various applications should execute while keeping a battery charged longer. Power consumption is also a major concern in server infrastructures where the amount of power consumed tend to be significant [44]. Moreover, power consumption of a System on a Chip (SoC) represents a limit to its integration capacity [31, 70].

All components of a MPSoC (e.g., processing resources, communication resources) participate to its total power consumption. In [64], it was predicted that the communication architecture will contribute by a larger portion to the total circuit power consumption. In [15], the authors report that this contribution can reach up to 50% in current and future devices. The importance of early power-aware DSE for communication architectures in MPSoC design was emphasized in [46].

Architecture cost

The monetary cost of the deployed architecture is also a quite recurring design objective. It can be either expressed in the problem model explicitly, such that each component is associated with a monetary cost (e.g., [38, 51]), or as parameters that may influence (increase or decrease) the cost like the number of components used in the architecture (e.g., [76]) or the memory usage (e.g., [39]).

2.1.6 Mapping and scheduling models

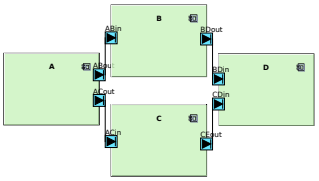
This Section first overviews different strategies to compute a mapping and scheduling solution depending on when mapping and scheduling decisions have been taken (referred to as scheduling strategies in the terminology in [61]), then, in later paragraphs, we will discuss other properties to categorize scheduling solutions.

Static/dynamic scheduling strategies

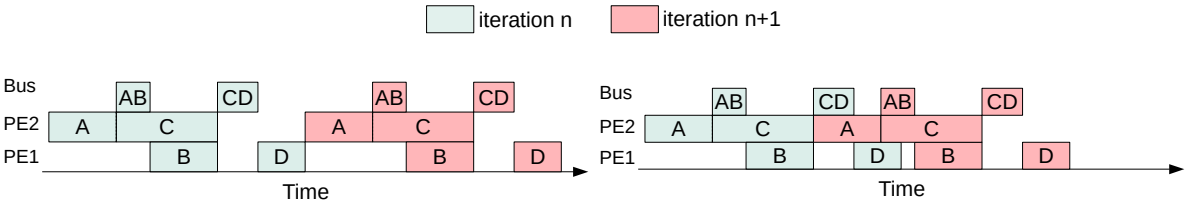
Scheduling strategies can be classified by how many decisions have been taken at design-time and at run time. We rely on the classification from [61] where six scheduling strategies have been identified. **Fully static** (also referred to as design time in this thesis) schedule means all decisions of scheduling task and communication, including the exact start times are taken at design-time. **Ordered transaction** schedule only determines the order of execution of tasks and communications but not their exact starting times. The **self-timed** schedule means only the order of execution of tasks is defined at design-time. **Quasi-static** scheduling provides multiple input-dependent schedules. Input-dependent schedules are useful when application execution depends on its input. For example, based on a statistic profiling of the execution of an application under different inputs, different scheduling solutions can be proposed. **Static assignment** scheduling only determines the mapping of tasks to processing resources at design time. The remaining decisions are left to run time. For **fully dynamic** schedules, all design decisions are made at run time for maximum flexibility, but also at the cost of the highest run time overhead.

Task preemption and migration

Preemption refers to the possibility to start the execution of a given task on a given resource, interrupt it before its completion, and resume it later on the same resource. **Migration** allows to resume it on another resource. Preemption and migration are useful in several scenarios. For instance, the preemption of a certain task t_1 allows another task t_2, with a higher priority, to take control of the resource that was allocated to task t_1. By multiplying the scheduling and/or mapping possibilities, preemption and migration may introduce significant overhead to DSE.



(a) Example of a directed acyclic graph



(b) Non-pipelined

(c) Pipelined

Figure 2.1: Pipelined and non-pipelined scheduling models

Pipelining in scheduling periodic applications

Periodic applications run *periodically* such that tasks of the application graph execute, in an iterative fashion, on different data sets. An *iteration* refers to the execution of all tasks of the application graph. The scheduling is called pipelined or non-pipelined depending on whether the iterations are allowed to overlap or not. In a **pipelined** scheduling, there is an overlap between tasks of two different iterations as illustrated in Figure 2.1c, for the application graph in Figure 2.1a. For a **non-pipelined** scheduling, illustrated in Figure 2.1b, tasks of an iteration $m + 1$ are not allowed to start before the completion of all tasks of iteration m.

2.2 Design Space Exploration Techniques

The search for a design solution with respect to one or several design objectives generally considers two orthogonal components [27]: 1) the evaluation of a single design point, and 2) the search strategy used to cover the design space during the DSE process.

2.2.1 Evaluation methods

In order to find a design solution which satisfies the design constraints, evaluation methods are required to evaluate a single design candidate. Evaluation methods can be organized in three main categories as described in [47]: 1) measurements on a **prototype implementation**, 2) **simulation**-based evaluations, and 3) estimations based on **analytical models**. Each category provides a different trade-off between exploration cost (e.g., run-time) and accuracy.

Measurement on a prototype provides the most accurate evaluation metrics, but may require a high exploration cost and a high development cost prior to exploration. Simulation techniques provide the a more balanced trade-off between accuracy and exploration cost compared to prototyping. Simulations can be performed at different levels of abstraction (e.g., Register Transfer Level (RTL), cycle-accurate, Transaction Level Modeling) as explained in details in [47]. Analytical models-based methods have lower accuracy than simulation-based methods, but are faster. Analytical estimations and simulation can be used in a complementary way, such that analytical models operate on a large design space to identify promising solutions (or parameters that lead to promising solutions), on which simulations are performed later for a more accurate evaluation.

2.2.2 Search strategies

Searching a design space towards valid, near-optimal or optimal solutions with respect to one or several design objectives, is a key component of DSE. Search strategies can be roughly classified into two groups: 1) Exact search, and 2) Approximate search [47]. Typically, a search strategy provides a certain trade-off between exploration effort and optimality. While exact search methods are able to guarantee the quality of the returned solution, approximate (also called sub-optimal) search methods trade optimality for a reduced exploration cost.

Approximate Search for DSE

Approximate search methods consist of heuristics and meta-heuristics that aim at finding good quality solutions fast. These methods tend to consider only a subset of design candidate solutions (i.e., incomplete search), thereby, are suited for large

design spaces, and for run time environments where the exact search is not reasonably feasible. However, there is no guarantee on the quality of solutions: there is no guarantee to find a solution when one exists, or that the solution is close or not to the exact optimum, and basically, a local optimum (i.e., a solution which is optimal within a limited neighborhood of candidate solutions from the design space) is found rather than the global optimum. Examples of known meta-heuristics include hill climbing, tabu search, simulated annealing, ant colony and Evolutionary Algorithm (EA). Further details on these methods can be found for example in [47].

Exact Search for DSE

Exact search basically relies on mathematical models (formulations) to describe a decision problem. A mathematical formulation refers to a mathematical description of a decision problem using a set of constraints over variables. Variables are defined on a certain domain. Constraints define relationships between variables. One or several objective functions defining the system metrics to minimize or maximize can also be considered. The satisfy design objectives (e.g., satisfy a threshold constraint on latency or throughput as explained in subsection 2.1.5) are described using constraints (e.g., $throughput \leq threshold_{throughput}$). These formulations generally constitute an input to a generic resolution engine called a solver. The latter examines the problem and returns solution(s) that satisfy the mathematical formulation. Formally, we refer to the mathematical formulation as a Constraint Satisfaction Problem (CSP), Definition 2.2.1 [48]. The definition of a solution to a CSP is given in Definition 2.2.2 [48].

Definition 2.2.1 (CSP)**.** A CSP P is a triple $P = \langle X, D, C \rangle$, where X is an n-tuple of variables $X = \langle x_1, x_2, \ldots, x_n \rangle$, D is a corresponding n-tuple of domains $D = \langle D_1, D_2, \ldots, D_n \rangle$ such that $x_i \in D_i$, and C is a t-tuple of constraints $C = \langle C_1, C_2, \ldots, C_t \rangle$. A constraint C_j is a pair $\langle R_{S_j}, S_j \rangle$ where R_{S_j} is a relation on the variables in $S_i = scope(C_i)$.

Definition 2.2.2 (CSP solution)**.** A solution to the CSP P is an n-tuple $A = \langle a_1, a_2, \ldots, a_n \rangle$ where $a_i \in D_i$ and each C_j is satisfied in that R_{S_j} holds on the projection of A onto the scope S_j.

CSPs can be classified according to the nature of the mathematical formulation (e.g., linear/non-linear, logical) and the techniques deployed to solve them.

Mathematical Programming Mathematical programming models are typically classified according to the types of the decision variables, constraints, and the objective function. Linear Programming (Linear Programming (LP)) is a mathematical programming technique where the problem is modeled in terms of linear inequalities, linear objective function, and where variables have continuous domains. To represent decisions of a discrete nature (e.g., yes, no, 0, 1, 2), Mixed Integer Linear Programming (Mixed-Integer Linear Programming (MILP)) problems can include both discrete and continuous variables. If all variables of the LP problem are integers, the problem is called an Integer Linear Programming (Integer Linear Programming (ILP)) problem. Simplex, interior point, Branch-and-Bound and cutting plane techniques are well-known algorithms used to solve mathematical programming problems [23].

Constraint Programming Constraint Programming (CP) defines a paradigm to solve combinatorial problems with and without optimization criteria. The problem to solve is first defined as a Constraint Satisfaction Problem (CSP). With respect to mathematical programming models, a CSP offers more flexibility to model logical constraints and arithmetic expressions (e.g., modulo, integer division, minimum, maximum). CP has no limitation on the arithmetic constraints that can be set on decision variables and can use ad-hoc constraints (e.g., the ”all-different” constraint), to accelerate the search of frequently used patterns [29]. CP has proven high efficiency in solving scheduling problems [29]. A search engine which typically relies on methods like backtracking and constraint propagation [48], is used to solve the CSP.

Boolean Satisfiability / Satisfiability Modulo Theories Satisfiability consists in the problem of deciding if a certain formulation that expresses a constraint has a solution. The basic satisfiability problem, denoted Boolean Satisfiability Problem (SAT), is expressed using Boolean variables related by logical connectives ($\wedge$, $\vee$, $\neg$, $\Rightarrow$, $\Longleftrightarrow$) and forming a formulation in *Boolean logic*. A valid solution to a SAT problem is an assignment of $true/false$ values to all Boolean variables such that the overall formula is true. However, a majority of real-world problems needs a more

expressive language to be described. *First-order logic* provides a very natural way to express problems. It include logical connectives ($\wedge$, $\vee$, $\neg$, $\Rightarrow$, $\iff$), variables, quantifiers ($\exists$, $\forall$), functions (e.g., $f(x), x + y$), predicate symbols (e.g., $x < y$) and constant symbols (e.g., $0, \pi$). Satisfiability Modulo Theories (SMT) relies on background theories to interpret a formulation in first-order logic. For instance, the theory of arithmetic defines the semantic interpretation of symbols like $+$, $\leq$, $>$, 0 or 1. A solution to a problem formulation in SMT is called a *model*. A *model* is an interpretation of variables, functions and predicate symbols, with respect to used theories, such that the overall formulation is *true*.

In view of the need to solve more and more complex combinatorial problems, researchers working on SAT and SMT have put many efforts in recent years to innovate in algorithms, data structures and heuristics [20]. Recent SAT solvers can check formulas with hundreds of thousands variables and millions of relationships between these variables [20]. SMT solvers are also following the same trend especially for the more commonly used theories [20].

Discussion The different classes of CSP each have their own characteristics and advantages, making them suitable for different kinds of decision problems. For instance, some design constraints can be naturally expressed by a linear inequation. Let us assume the following mapping problem model: x_{tp_i} is a Boolean variable that denotes the mapping of a task t to processor p_i and x_{tp_j} is a similar variable that denotes the mapping of the same task t to another processor p_j. A design constraint is to make sure that task t is assigned to at most one processor. This means that task t is either mapped to processor p_i (i.e., $x_{tp_i} = 1$), or to processor p_j (i.e., $x_{tp_j} = 1$). This constraint can be naturally expressed by the following linear inequality: $x_{tp_i} + x_{tp_j} \leq 1$. Now, let us consider a scheduling problem, where a design constraint is to ensure that two different tasks t_i and t_j cannot overlap in time on a given processor p. This constraint requires a disjunction of constraints and cannot be expressed naturally by a linear inequality. In fact, mathematical programming models support only linear constraints, linearized logical constraints, or quadratic convex constraints [29]. Instead, a first-order logic model allows to capture such a disjunctive constraint naturally using logical operators (e.g., $((x_{t_ip} = 1) \wedge (x_{t_jp} = 1)) \Rightarrow ((start_{t_i} > end_{t_j}) \vee (start_{t_j} > end_{t_i}))$.

The main advantage of the methods discussed in this subsection is their **completeness**, meaning their ability to guarantee optimality of the returned solution, to find a solution if one exists, or to prove unfeasibility. However, the search for optimal solution can come at a high cost. Several techniques are commonly used to **address the complexity of exact search** methods. These techniques can roughly be classified into two major groups of approaches: 1) decomposing the problem into smaller sub-problems, and 2) navigating the search smartly during DSE. The first approach deals with large problems by decomposing them into several smaller sub-problems, and then merging sub-problems results. However, ignoring interdependecies between sub-problems may lead to sub-optimal solutions. The second approach uses problem-specific knowledge to prune the design space (i.e., eliminating unsuitable design candidates) and thereby, improves the efficiency of the search. For example, Bonfietti et al. [9] explore this latter direction and result in one order of magnitude improvement of DSE runtime. The major challenge for these techniques remains to restrain the impact on completeness.

2.3 Targeted Systems

In this thesis, we target off-line Design Space Explorations (DSEs) that are performed at design time. To capture applications and their execution semantics, we rely on the **SDF** MoC for its expressiveness and high analyzability at design time. Specifically, we use AHSDF [50] for the most parallel representation of one iteration of the SDF graph. We consider multiple applications by means of **separate independent graphs**. These applications can have **possibly different latency objectives**: e.g., satisfy different real-time deadlines for each application, satisfy real-time deadlines for several applications and minimize latency for other applications. This allows to study a workload of applications with different timing design objectives.

We consider an architecture model where a set of processing resources are interconnected according to a **segmented bus (multi-bus)** communication model. Each bus segment can have a different bandwidth capacity and the topology of interconnection is not constrained (e.g., tree, chain, ring). The processing model is not constrained as well: it can be **homogeneous**, **weakly heterogeneous**, or **strongly heterogeneous**. However, we are mainly interested in heterogeneous architectures.

To ensure efficient use of architecture resources, design decisions of both **task and communication mapping and scheduling** are regarded in an integrated way. The expressiveness and high analizability of SDF allow for static scheduling. A **fully static** scheduling solution is created. This scheduling solution can be used entirely, or partially in the context of a more dynamic scheduling strategy, like ordered transaction, self timed and static assignment. In reality, there may be variations in defined execution times, therefore, when implementing the scheduling solution, it is possible to preserve only the order of tasks and/or communications. This is a common practice to generate schedules for dataflow graphs, see for example [61]. Moreover, fully static schedules are still very useful to provide performance guarantees which are crucial for real-time systems. For example, scheduling solutions which may violate real-time deadlines are identified and eliminated at design time and so, dynamic schedulers can focus only on viable design solutions. **Preemption** and **migration** of tasks are not allowed. If an application is periodic, multiple successive iterations can be modeled as independent applications, to allow for a **pipelined scheduling** or a **non-pipelined scheduling**. Precedence relationships and data exchanges(if existing) between different iterations of the same application can be specified. This is of course limited since it only allows to analyze a finite and fixed number of iterations but can be useful.

In next Section, we discuss most relevant related work, with regard to our work, and to the criteria presented in Sections 2.1 and 2.2.

2.4 Existing Design Space Exploration Work

In this Section, we summarize and discuss the work most closely related to ours:

Bonfietti et al. [9] were the first researchers to demonstrate that complete search techniques are practically viable for the mapping and scheduling of SDF. In their early work [10], the authors proposed a CP formulation for the joint mapping and scheduling of SDF actors, onto a homogeneous multi-processor architecture, under minimal throughput design objective. The proposed scheduling model is self-timed. Here the communication-oblivious model of communication is adopted. The efficiency of the proposed DSE approach was tested on HSDFs (up to 25 nodes) generated by the SDF3 generator [63] and an architecture with 2, 4 and 8 processing resources using

ILOG Solver 6.3. Results showed that solver run-time grows exponentially with the size of the instances. For graphs counting 10 to 20 nodes, a solution can be found in less than 20 minutes, which is reasonable for a design time DSE.

The work in [9] extends the formulation proposed in [10] by a series of acceleration techniques that aim at improving the search efficiency and reducing the solver run-time. The authors introduce several reduction techniques that define tighter bounds of the problem formulation by: 1) prioritizing the mapping of the most impacting actors[1], and 2) tending to map actors related by a data dependency on the same processing resource. However, as opposed to the reductions that we will present later in this manuscript, their reductions do not preserve optimality. [11] further extends prior work by proposing a novel formulation for the throughput constraints in [9, 10]. This novel formulation is called "incremental" as it avoids the recomputations of the throughput when partial solutions are found.

Experiments showed that the proposed reduction techniques in [9] reduced the solver run-time by one order of magnitude, at the cost of 22% on average loss of optimality. The authors also attempted to reduce the solution run-time by constraining the mapping of all the HSDF tasks corresponding to repetitions of the original SDF actor to the same processing resource. This limitation of a degree of freedom in the mapping led to around 30% optimality gap in throughput. In our work, we allow tasks to be mapped freely on available and compatible processing resources. Also, we account for communications on a bus-based communication architecture, while here communications are considered instantaneous. Another difference with our work is that, here, only a single application graph is considered rather than multiple independent applications. Besides, only throughput is considered in [9, 10], while in our work, we consider latency and power consumption.

Rosvall and Sander [50] propose a CP-based DSE framework for the mapping and scheduling of dataflow applications—modeled as AHSDFs—on multiprocessor architectures, with local memories. Similarly to our work, the output scheduling can be fully static or self-timed. Also, multiple independent application graphs are considered simultaneously as well. The work in [50] accounts for mixed-criticality applications. Critical applications are first deployed to respect some hard constraints

[1]The impact of an actor here is measured based on its execution time and on the actors that depend on its execution.

on performance (e.g., a threshold on throughput), while non-critical applications are deployed to remaining resources to optimize throughput, with best-effort. This can be achieved in our work as well.

Here, the single shared bus communication model is adopted such that a number of TDMA bus time slots is allocated to processing resources that send data over the shared bus. The developed framework derives performance analysis information in terms of throughput, memory consumption, and processor utilization. The execution of different iterations of an application is allowed to overlap in time (i.e., pipelined scheduling model). As a search strategy, this work relies on a constraint solver to meet satisfy design objectives and a complementary branch and bound algorithm to find solutions optimized for throughput.

In [35], the contribution in [50] is enriched by addressing feedback control tasks in addition to dataflow applications. Feedback control tasks are modeled using the periodic task model. AHSDF actors are scheduled at design time similarly to [50]. Periodic tasks are scheduled at run time according to a fixed priority algorithm. The main design objectives for the exploration are throughput for AHSDFs and deadlines for periodic tasks.

For evaluation, the authors in [50] propose to run experiments on four real-world streaming applications and a target platform consisting of 8 homogeneous processing resources, connected via a shared bus. All possible combinations of the four applications sharing the platform are analyzed (upto 32 tasks and 36 communications). The design objectives were set to optimize (maximize) throughput for one of the four applications, while satisfying a threshold on throughput for each of the remaining applications. Experimental results showed that solutions for a satisfy design objective were found in the order of seconds for all studied workloads. However, for 6 out of 15 workloads, the search for optimal solution could not be completed within a time limit of 30 minutes.

The framework is further extended to study the trade-off between throughput and processing-related power consumption in [51]. Techniques to improve scalability are proposed as well. A two-step approach is introduced such that a heuristic function is first applied to find good quality solutions fast and the complete CP search is used as a second step to prove optimality of the first step solutions. The first step uses additional constraints to attempt to reduce the search space around promising can-

didate designs. These constraints aim at: 1) forbidding the sharing of processors between applications, and 2) forcing the number of available processors that an application is allowed to use to the (estimated) least number of processors necessary to match the required throughput. Here, problem-specific knowledge is exploited to speed-up the DSE procedure, as we will propose as well later in Chapter 4. The second step removes the constraints on the mapping, and the best solution found in the first step is used to constrain throughput. The two-step approach demonstrated better efficiency compared to the single-step approach [50] in terms of optimal solutions search run-time. However, when introducing a hardware accelerator to the architecture, the search run-time grows dramatically (6 hours were not sufficient to reach throughput optimal solutions for 4 out of 15 workloads).

Contributions in [35, 50, 51] represent a practical evidence of some properties of a CP-based approach, namely: 1) extensibility by adding new design decisions and objectives [35, 51], without changing the existing model, and 2) applicability to solve holistically DSE problems. However, in either contributions, only a single shared bus architecture is supported, and the power consumption analysis disregards power consumed by the bus. Our work aims at addressing these limitations, by accounting for various multi-bus topologies, and minimizing the interconnect power consumption.

The work in [49] builds on the contributions in [50] and considers both the interconnect and the processors power consumption. However, it targets platforms with a Temporally Disjoint NoC (TDN-NoC) with a mesh topology. In [49], the authors focus on TDN-NoCs that use deflective routing: packets are never buffered in switches. This makes the modeling of TDMA buses amenable to capture the traffic on links in these TDN-NoCs. Similarly to TDN-NoCs, we do not consider buffering in inter-segment bridges. In addition to the mapping and scheduling, the DSE problem that the authors in [49] study includes finding a conflict-free allocation of communication slots in the TDN-NoC. In this problem, the authors consider the routing strategy to be fixed by the network, whereas in our work all possible routes are explored between a producer and a consumer processing resources. This work has the advantage of extending the considered communication architecture from a bus-based architecture [35, 50, 51] to a NoC-based architecture [49]. It is again a demonstration of the extensibility of the CP approach. Such an extension can represent a perspective to

our future work.

The work of Tendulkar et al. [66–68] targets homogeneous many-core processor platforms (e.g., Kalray MPPA-256 [69]) where mapping and scheduling of dataflow application graphs are studied, relying on a SMT approach and a complementary binary search. The target communication model is a NoC, where processors are grouped into clusters, and communicate with each other via Direct Memory Access (DMA). Communication inside a cluster is performed by means of a shared memory and is assumed negligible. In this work, an accurate modeling of a DMA mechnism is provided in order to characterize the data transfers between different clusters. However, different routes are not explored during mapping. In terms of design objectives, the contributions in [67] are focused on latency and buffer size, whereas we consider interconnect power consumption along with latency. Similarly to our work, task preemption is not allowed. The proposed scheduling is non-pipelined. The precision of the SMT modeling was evaluated against measurements of latency on a real many-core hardware. The maximum observed error in terms of latency was 27%. The authors explain that this error is mainly due to ignoring contentions on the interconnect. In our work, we address this limitation by accounting for contentions during scheduling and communication latency estimations.

To reduce the run-time of the DSE process, the work in [67, 68] exploits the symmetries present in the architecture and the application. Architecture-related symmetry constraints aim at forcing one specific mapping when multiple alternatives exist that only differ in terms of the permutations of pairs (task, processing resource). Application-related symmetry constraints aim at simplifying DSE when, for instance, task dependency graphs are symmetrical in terms of topology and of the characteristics associated to tasks (e.g., in a HSDF, tasks corresponding to different firings of the same SDF actor). In this case, a lexicographic scheduling order is forced instead of exploring all identical permutations of all scheduling orders.

The symmetry breaking techniques were evaluated on a set of applications from StreamIt benchmark [71] and a homogeneous architecture where 5 and 20 processors are deployed in turn. Results showed that for the 5-processor case, the size of application graphs, for which optimal solutions were found within the time out of 20 minutes, is increased from 12 to 48 tasks. Results also showed that the processor symmetry breaking benefits increase with the number of processors deployed, which

is expected as symmetries become more important. Unfortunately, in our work, the use of symmetry breaking techniques is considerably limited by the heterogeneity of the processing model (e.g., CPUs and specialized DSPs) and the irregular topology and bandwidth capacities of the multi-bus communication model. However, we reuse symmetry breaking for application tasks as in [67].

A run time manager complements the contribution in [67] to optimize resource-utilization. It is in charge of taking decisions for dynamic task migration, at a predictable time and without affecting other applications running on the platform. Such a run time manager can be also used in our work to improve the awareness of behaviors that are hardly predictable at design time.

Lin et al. [38] propose a series of search algorithms relying on ILP and EA to solve the problem of mapping and scheduling of real-time SDFs on bus-based architectures. Generated output consists either of a single solution or a Pareto front for the optimization of throughput, latency and architecture cost. Studied architectures are heterogeneous and can include multiple buses and both local and shared memories. The proposed search strategies are 1) a global ILP model, 2) a two-stage ILP (first ILP for mapping followed by a second ILP for scheduling), and 3) a scalable mixed approach that integrates an EA with ILP. The output solutions map the SDF graph to processing and communication resources, and propose fully static schedules on processors and buses. Two possible routing policies are considered on buses: direct communication between processors and two-step communication via the shared memory. This work has the advantage of considering communications through both shared and local memories. However, in either case, it is assumed that a communication step involves only a single bus. In fact, despite the consideration of multiple buses in the architecture, this work assumes a fully-connected communication model, such that the shared memory and all processors are accessible through any bus. Processing resources are then allowed to communicate with each other through dedicated channels. On the contrary, in our work, we support a multi-bus communication model, where buses can be disposed in an ad-hoc topology between processing resources and data transfers between processing resources can cross as many buses as needed, we analyze the routing scheme, and account for possible contentions on the interconnect.

The efficiency of the approach [38] was evaluated using a set of graphs (5 to

15 actors) randomly generated by SDF3 [63], and a 3-processor platform, assuming a communication-oblivious model and a single bus model. For the first model, the single step ILP approach showed run-time values increasing up to 20 minutes for the 15-actor graph, while the two-stage ILP and the mixed approach showed both lower run-times: around 100 seconds for the 15-actor graph. Some experiments were also conducted on a MP3 decoder to evaluate the efficiency of the mixed EA-ILP approach in generating Pareto optimal solutions. The search lasted 23 hours when communications are assumed negligible, and 41 hours when they are analyzed. In this work, the applicability of the reduction technique we propose in Chapter 4 can be investigated to improve run-time.

Voss et al. [75, 76] propose a joint mapping and scheduling of tasks and communications to a multi-core architecture with a shared memory. The work of Voss et al. [75] is similar to ours in multiple ways: solving the mapping and scheduling of tasks and communications in an integrated way, using of a discrete time line (i.e., time slots) to determine start times of tasks and communications and producing a static schedule, relying on SMT and complementary search techniques (e.g., binary search), and integrating their work to a model-based tool for the development of embedded systems. The design objectives here consist mainly on giving guarantees on latency (i.e., satisfy and minimize as explained in Section 2.1.5). The usage of resources is also optimized (e.g., minimize number of processors used). Power consumption is not considered though. The proposed approach accounts for a safety criteria: tasks and resources are assigned with different criticality levels, such that higher-level tasks cannot be assigned to lower-level resources during mapping. This can be achieved in our work as well by enabling the user to add ad-hoc constraints, but it is not the focus of our work currently.

From the viewpoint of the communication architecture, it is not clear if their work supports multi-bus architectures as all examples display a single shared bus. The authors demonstrated that state-of-the-art SMT solvers can be used to efficiently and jointly compute a mapping and a scheduling of tasks and communications.

This work is focused on the domain of automotive systems. The approach was evaluated using randomly generated application graphs (in form of a DAGs) consisting of 5 up to 70 tasks. Results showed that the solver run-time grows exponentially. Up to 35 tasks, optimal latencies were reached in less than 30 minutes, while for the

graph with 70 tasks, exploration lasted around 29 hours. This work could investigate the use of our reduction technique, presented in Chapter 4, to tighten the scope of definition of time slots related to the scheduling of tasks and communications. This may result in significant improvements of the DSE run-time.

A representative body of works on several design issues of a segmented bus architecture called SegBus is that of Seceleanu et al. [54–58]. In the context of our research, the work described in [55, 56] is the most relevant: it describes a resource allocation methodology for SegBus. Here, several design issues of a segmented bus architecture are addressed. The aim is to find a mapping of components (i.e., master and slave components) to bus segments and a linear organization of bus segments, such that inter-segment traffic is minimized and parallel traffic on different segments is maximized. With respect to our work, this contribution does not consider power consumption of bus segments. Here, the topology synthesis is part of design decisions. However, a topological restriction is that bus segments can be only linearly connected: more complex topologies (e.g., ring, tree, ad-hoc) are not studied. As such, no routing is needed since there is a unique linear route between each pair of components. Moreover, the mapping of tasks to processing resources is left for future work, while a predefined communication matrix defines communication traffic between each pair of architecture components. This approach could be used combined with our DSE to find a fully motivated decisions of topology, mapping and scheduling of tasks and communications for linear multi-bus architectures.

As a search strategy, Seceleanu et al. [56] proposed heuristic methods and an exact search algorithm to find the linear segmentation and component-to-segments assignment. The authors report that exact search is a feasible solution since, in practice, the number of bus segments is rather modest. The proposed heuristic methods are articulated in two algorithms, the first algorithm aims at finding the component-to-segments allocation. It relies on an initial randomly selected component-to-segments allocation, then performs a greedy local search to generate improved solutions. Once optimized component-to-segments allocations are returned, the second algorithm performs random swap/move operations for a given component-to-segment allocation. Seceleanu et al. show that the heuristics find solutions as good as the optimum for problem instances where up to 4 bus segments are considered. However, for larger problem sizes, the distance from the optimum is not provided.

Table 2.1: A summary of selected related work

Ref.	TM[2]	TS[3]	CM[4]	CS[5]	Communication model	Design objectives[6]	Processing Model	Search Strategy
[10]	✓	✓	×	×	communication-oblivious	Thr	homo.	CP
[9]	✓	✓	×	×	communication-oblivious	Thr	homo.	CP + Opt
[50]	✓	✓	✓	✓	shared-bus	Thr	heter.	CP
[35]	✓	✓	✓	✓	shared-bus	Thr $\wedge$ L	heter.	CP
[51]	✓	✓	✓	✓	shared-bus	Thr $\wedge$ Pw		CP + Opt
[49]	✓	✓	×	✓	TDN-NoC	Thr $\wedge$ Pw		CP
[66–68]	✓	✓	×	✓	NoC	L	homo.	SMT + Opt
[38]	✓	✓	✓	✓	fully-connected	L $\wedge$ C $\wedge$ Thr	heter.	ILP+ EA
[75, 76]	✓	✓	✓	✓	shared-bus	L $\wedge$ C[7]	heter.	SMT
[55]	×	×	×	×	multi-bus	L	–	Exact + Heuristics
Our work	✓	✓	✓	✓	multi-bus $\wedge$ shared-bus	L$\wedge$ Pw	heter.	SMT + Opt

2.5 Summary and Conclusion

Design Space Exploration (DSE) is a wide research topic that encapsulates a large spectrum of problems depending on design decisions, objectives, target architecture, etc. We have identified and discussed the different components of a DSE approach. These range from the specification of the DSE problem (Sections 2.1) to the techniques (Sections 2.2) deployed to solve it and find sought design solutions. This study, together with a review of existing works (Section 2.4), allowed us to notice a

[2]Task Mapping
[3]Task Scheduling
[4]Communication Mapping
[5]Communication Scheduling
[6]Thr: Throughput; L: Latency; Pw: Power consumption; C: cost.
[7]Cost here refers to the number of architecture resources used.

lack in DSE research work that target the multi-bus communication architecture. As discussed in Section 1.1.4, the multi-bus communication architecture offers multiple advantages and is widely used.

In this thesis we provide a DSE approach for the mapping and scheduling of tasks and communications on a multi-bus architecture. We rely on a **Constraint Programming (CP)** approach, specifically based on **Satisfiability Modulo Theories (SMT)**, to tackle the DSE problem in a holistic way. In the last decade, in parallel to a considerable improvement of solvers efficiency, CP gathered much interest in the DSE research community where many works [9, 11, 35, 49–51, 67, 75, 76] demonstrated its applicability to holistically solve mapping and scheduling problems. In fact, while most approximate strategies decompose the DSE problem into a mapping step followed by a scheduling step (see for example [8, 22, 43, 62]), the CP allows to construct a holistic view on design decisions, and thereby, prevents the inherent loss of optimality due to decomposition. Furthermore, in the literature, there is no exact solution for the problem we are studying (mapping and scheduling of tasks and communications on a multi-bus architecture). Therefore, starting with a development of an exact method is a natural choice since conceiving approximate methods requires to dispose a priori of an exact method to evaluate the quality of solutions found by the approximate search. Thus, the work of this thesis can be the basis to design and calibrate faster approximate methods in the future. Moreover, in approximate strategies, the problem definition and the search engine are tightly coupled. Therefore, changing the problem definition, by adding new performance metrics, or extending to new communication architecture models for example, may make the search algorithm unviable. Hence, such changes would require adjustments of the heuristic search algorithm or meta-heuristics parameters. On the contrary, the CP approach is based on a separation between problem definition and the search engine. This separation of concerns results into a modular DSE, where there are more opportunities to extend the problem definition module with new specifications without changing the search module, and often without modifying the existing model (see for example [35] with respect to [50]). Furthermore, the problem definition module can be reused—partially or entirely—with a new search module, as new advancements are continuously integrated within search engines. As an example, the modularity and extensibility properties facilitate the extension of our work to support other communi-

cation architecture models (e.g., NoCs) by only adjusting constraints that govern the scheduling of communications (e.g., to account for packetization and depacketization in NoCs). Another advantage of the CP approach is that the search can guarantee optimality for a range of problems, and provide a trade-off between the search cost and the quality of solutions for larger scale problems.

We specifically rely on a SMT approach for the following reasons. First, based on the first-order logic, the CSP specified in SMT benefits from a high level of expressiveness, which is appreciated in the DSE problem we study. For example, scheduling on a resource-constrained architecture needs a disjunction of formulas to be captured properly. In fact, scheduling tasks on a set of constrained processing resources is a non-convex problem, which requires *disjunctive* constraints: To express that a processing resource can only execute a task, at a given time, the CSP should indicate that, if two tasks t_1 and t_2 are assigned to the same processing resource, then, task t_1 can start either after task t_2 has completed *or* vice-versa. Expressed using logical operators in SMT, this relation can be solved using search techniques which are suitable for this kind of problems. However, LP, for example, is adequate when the problem is specified in terms of a conjunction of linear constraints and feasible solutions are in the space of a convex polyhedron.

To deal with the complexity of the holistic analysis and the complete search of the CP-based approach, we further propose a pre-analysis to **exploit problem-specific knowledge** (i.e., application graph and architecture resources). This analysis aims at **pruning the design space** to **improve the scalability** of the exact search by reducing its computational cost, **without impacting completeness**.

The contributions of this thesis differ from existing work presented in Section 2.4, and summarized in Table 2.1, mainly by three points: First, none of the discussed work treats the holistic mapping and scheduling on a multi-bus architecture and related design issues such as: where and when execute tasks and communications on a distributed bus architecture? How to manage inter-segment and local communications on multiple bus segments? How to manage different bandwidth capacities of bus segments? We will present our DSE proposal to address these issues in Chapter 3. Second, the reduction technique presented in this thesis is novel and original. Most significantly, it results in reducing the DSE run-time drastically—by few orders of magnitude on some of the studied workloads—without impacting completeness

(i.e., optimality, guarantee to find a solution if one exists, guarantee to prove unfeasibility), as we will showcase in Chapter 4. Third, none of the presented work allow to study both latency and power consumption related to the interconnect, nor to capture power consumption of a multi-bus interconnect at system-level. We will present this latter contribution in Chapter 5.

Chapter 3

A Satisfiability Modulo Theories Formulation for the DSE of Multi-Bus architectures

3.1 Introduction

The communication architecture of a MPSoC strongly influences its overall performance and power consumption [46].

As we argued in Chapter 2, there is a lack of Design Space Exploration (DSE) research work that targets multi-bus architectures. These, though, represent a solution to many SoC design issues such as performance, power consumption and Intellectual Property (IP) utilization, as reflected in [6,13,28], and discussed in Section 1.1.4.

In this Chapter, we propose an approach to explore design alternatives for the mapping and scheduling of tasks and communications on architectures with multiple buses. This approach relies on Satisfiability Modulo Theories (SMT) and it is our **first contribution** in this thesis. Our contribution is positioned at system-level of abstraction and targets the mapping and scheduling of dataflow applications (e.g., signal, video and image processing).

To deal with the complexity of the problem of mapping and scheduling tasks and communications, a common practice in related works consists in decomposing it into

sub-problems [8, 22, 43, 62]. This decomposition reduces the complexity of exploration but disregards interdependencies between the sub-problems, which unfits in particular the exploration of multi-bus architectures. In fact, unlike the shared bus where all pairs of components communicate with each other at the same bandwidth, in a multi-bus interconnect, buses can operate at different bandwidths, providing non-uniform communication schemes between architecture components. Between pairs of components, even if these components are identical, the cost of communication depends on how each pair of components can communicate with each other. For example, let's assume a first pair of processors P_{A1} and P_{B1}, and a second pair, P_{A2} and P_{B2}, such that P_{A2} is identical to P_{A1}, and P_{B2} is identical to P_{B1}. P_{A1} and P_{B1} are connected by a bus which bandwidth is equal to bw_1, and, P_{A2} and P_{B2} are connected by another bus which bandwidth is equal to $bw_2 = 2 \times bw_1$. The cost of communication between each pair of processors is therefore different. In this case, the overall latency (execution + communication) depends on task mapping, on communication mapping, and on the scheduling of tasks and communications. To capture how task and communication mapping and scheduling impact each other, we suggest to tackle this problem in an integrated way.

The remainder of this Chapter is organized as follows: Section 3.2 defines the problem addressed in this Chapter. Sections 3.3 and 3.4 present respectively the properties of the application model and the target architecture that are considered as an input to our DSE. Section 3.5 describes the deployment solution that represents the output of the exploration and the related assumptions. Section 3.6 provides the details of our SMT formulation and explains all the variables and constraints. In Section 3.7, we present the implementation of our approach and its integration into a model-based design tool. A set of experiments are presented in Section 3.8 to illustrate a case study where our approach is used to guide the design of new architectures. This can be achieved by comparing the found optimal solutions for various architectures candidates, and selecting the architecture that best meets the design objectives for given workloads. Finally, Section 3.9 summarizes this Chapter, discusses limitations of the proposed approach and gives a glimpse into the second contribution of this thesis which will be presented in the following Chapter.

3.2 Problem definition

In this Section, we define the problem to solve in this Chapter, as follows.

- Given
  - A workload containing one or several dataflow applications;
  - A set of processing resources interconnected in a predefined topology with a multi-bus interconnect;
  - A set of latency design objectives. A *latency design objective* can be specified, *for each application from the workload*, among the two following alternatives[1].
    - *Real-time design objective:* end-to-end latency is subject to a deadline constraint. The solution must ensure that latency of a given application meets a deadline constraint. A deadline is defined as a temporal threshold on the execution of a given application.
    - *Lowest-latency design objective:* The exploration procedure must select the solution which provides the lowest end-to-end latency among all candidates of the design space, with or without a deadline constraint;
- Find:
  - A mapping of tasks and communications to processing and communication resources;
  - A scheduling of tasks and communications on processing and communication resources.

3.3 Workload Model

We focus on dataflow applications that are typical of signal, image and video processing systems. A dataflow application consists of several units of work (tasks) that represent the computations necessary to process data streams. Tasks are related to

[1] Applications within a given workload can have different latency design objective.

each other's by data dependencies (communications). A data dependency defines a relation between two tasks such that the execution of the destination task depends on data produced by the source task. The task which produces data is referred to as a *producer* and the task which consumes it as a *consumer*.

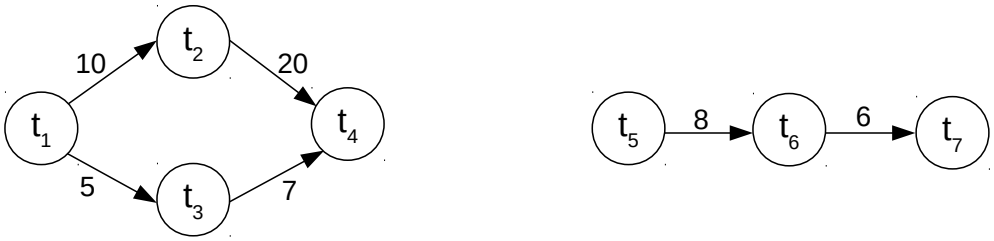
We denote an application by means of a Directed Acyclic Graph (DAG). This graph can correspond to a AHSDF obtained from the transformation of an original SDF graph, as explained in Section 2.1.1. As we consider **multiple applications** running in parallel and sharing architecture resources, we denote the set of all applications a *workload*. Each application A, from the workload, is captured by an **independent** DAG denoted as $G^A = \langle T^A, C^A \rangle$ (Figure 3.1). The definition of G^A is given in Definition 3.3.1.

Definition 3.3.1 (DAG)**.** A DAG G^A of an application A is a tuple $\langle T^A, C^A \rangle$ where T^A is a finite set of vertexes denoting application *tasks* and C^A is a finite set of directed edges denoting *data dependencies* (or *communications*) connecting tasks in T^A. Each data dependency c_{t_m,t_n} in C^A is directed from one task t_m (*producer*) to another task t_n (*consumer*), such that there is no path that starts at a vertex t in T^A, traverses a directed sequence of edges and loops back to t.

We denote the union of all workload tasks as $\mathcal{T} = \bigcup_{A \in \mathcal{W}} T^A$ and the union of all workload communications as $\mathcal{C} = \bigcup_{A \in \mathcal{W}} C^A$.

A producer is also called a *predecessor* of the consumer and the consumer is called a *successor* of the producer. A task with no predecessors is called a *source* (t_1 in Figure 3.1a), while a task with no successors is called a *sink* (t_4 in Figure 3.1a).

Each data dependency $c_{t_m,t_n} \in \mathcal{C}$ is annotated with a positive integer denoting the amount of data d_{t_m,t_n} that needs to be transferred from the producer task t_m to the consumer task t_n. Each task t is annotated with the *operation type* op_t (e.g., Fast Fourier Transform (FFT)) used to map the task to Processing Elements (PEs) capable of executing it. We denote $F(t)$ the set of PEs capable of executing task t. Tasks are also annotated with the set of their execution times on compatible PEs, $ET_t = \{et_{t,c}, c \in F(t)\}$. The sink is annotated with a positive integer denoting the application's deadline $deadline^A$. Examples of DAGs are shown on Figure 3.1 in which the op_t, ET_t and $deadline^A$ annotations are omitted.



(a) Example of an application A_1 DAG. (b) Example of an application A_2 DAG.

Figure 3.1: Examples of applications' DAGs. Vertexes represent tasks, edges represent data dependencies annotated with the amount of exchanged data.

Without loss of generality, in our models, the application graphs have each a unique source and a unique sink. If the modeled application has more than one source and/or sink, virtual source and/or sink vertexes are added. These virtual vertexes have zero execution times and produce (consume) data with a null communication cost.

Periodic applications are currently not supported. The only workaround consists in modeling successive iterations as one single application. This approach only allows to analyze a finite and fixed number of iterations, which is a strong limitation regarding periodic applications. Concerning dependencies between iterations of the same application, if data are exchanged between iterations, these exchanges can of course be modeled with regular edges. If there are no data passed from one iteration to the next, but we want to impose an order of execution among iterations, this can also be modeled with regular edges with zero amount of exchanged data. This approach is of course limited but it allows to explore portions of a periodic scheme, which can be useful.

3.4 Architecture Model

The architecture model is captured by an undirected graph G_{arch}. The definition of the undirected graph is given in Definition 3.4.1.

Definition 3.4.1 (Architecture graph)**.** The architecture graph G_{arch} is a tuple $\langle U, L \rangle$ where U is a finite set of vertexes that represent processing units (resources) or communication units (i.e., a bus or a bridge) and L is a finite set of edges that represent

links between them. Edges are only allowed to link bus units to units of other types (i.e., processing units, bridges).

A Processing Unit (PU) is modeled as a composition of one or several Processing Element (PE). Every PU p is annotated with its *operation set* O_p, and the size of *memory* $mem_p \in \mathbb{N}$ available to the unit. The operation set enumerates the list of operation types that can be performed by the unit. For a generic processor (e.g.,CPU), the list is empty to indicate that the unit is capable to execute any type of operation. When a PU is composed of multiple PEs, the operation set is equivalent for all PEs and is that inherited by the global PU. We don't model memories as separate units. The available memory to a PU is divided equally between its PEs. Each PE has access to a predefined memory region, spatially separated from other regions.

Vertexes that model bus units are annotated with the *bandwidth* $bw_b \in \mathbb{N}$. It measures the amount of data that can be transferred through the bus per time unit.

The definition of a route in the architecture is given in Definition 3.4.2.

Definition 3.4.2 (Route)**.** We call a *route* between a PU i and a PU j in the architecture graph, the set of communication units joined to each other by a sequence of edges, and to i on one end and j on the other end. A route contains n buses and $n-1$ bridges. It can be composed of a single ($n = 1$) bus segment and zero bridges, or multiple bus segments ($n > 1$), interfaced by bridges.

The following assumptions are considered for the target architecture model:

- *Assumption* 1: A PE executes only one task at a time.
- *Assumption* 2: In our modeling, each PU disposes of a local memory, shared between local PEs: PEs inside a PU communicate with each other via this memory. Tasks allocated to a PE, inside the PU, read (or write) data in this local memory. All communications inside a PU have a negligible overhead. This assumption is commonly used in similar work, see for example [49, 50, 67]. However, we will discuss later in this Section how this assumption can be relaxed to capture communications via an external shared memory.
- *Assumption* 3: Data transfers and task executions can run in parallel if they are not related (that is, if the transferred data is not produced or consumed by

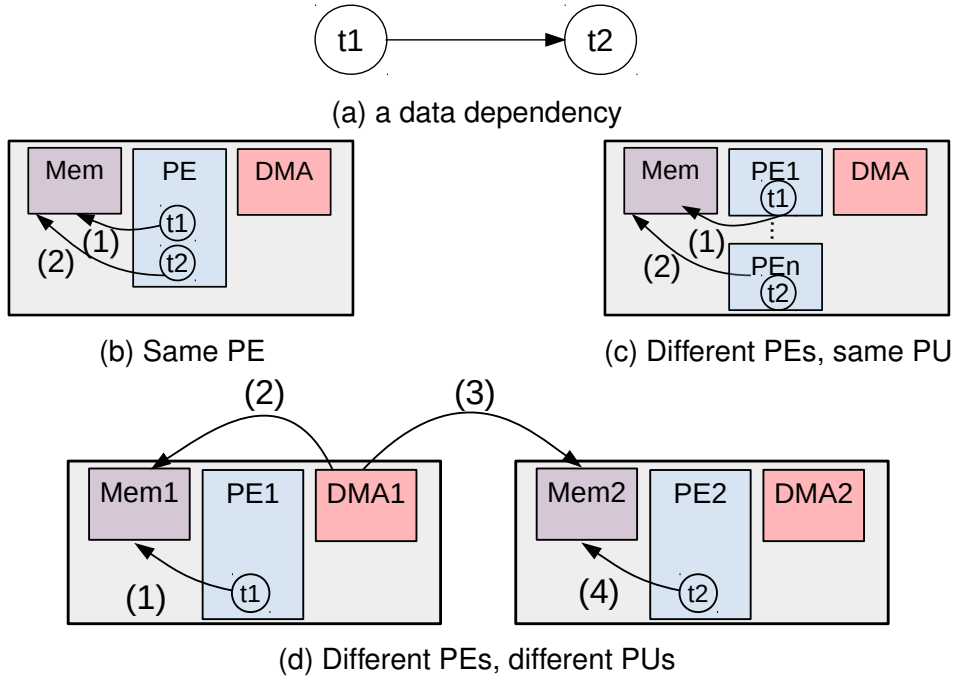


Figure 3.2: Communication mapping scenarios.

the running task). For example, a DMA engine can be used to perform data transfers in parallel to processing.

- *Assumption* 4: There are no deadlocks, livelocks or data losses on buses during a transfer.

3.5 Deployment Solution

The deployment solution provides a mapping of tasks to PUs (and to PEs), a mapping of communications to routes and a time-based schedule of tasks and communications on mapped resources. A communication can traverse a single bus segment or as many bus segments and bridges as needed to reach the intended destination. We consider the following assumptions for the mapping and scheduling.

- *Assumption* 5: Considered communication mapping scenarios are illustrated in Figure 3.2. The latter illustrates three possible communication scenarios for two tasks t_1 and t_2, related by a data dependency (Figure 3.2a).

In the first scenario (Figure 3.2b), the producer and the consumer of the data dependency are mapped to the same PE and thereby, as stated by *Assumption* 2, produce (arrow marked with (1)) and consume (arrow marked with (2)) data to/from the same memory. The communication time in this scenario is negligible according to *Assumption* 2.
In the second scenario (Figure 3.2c), the producer and the consumer are mapped to distinct PEs, but inside the same PU. The communication scenario here is similar to the one in the first scenario. All communications are done within the same PU, therefore, communication time is negligible according to *Assumption* 2 as well.

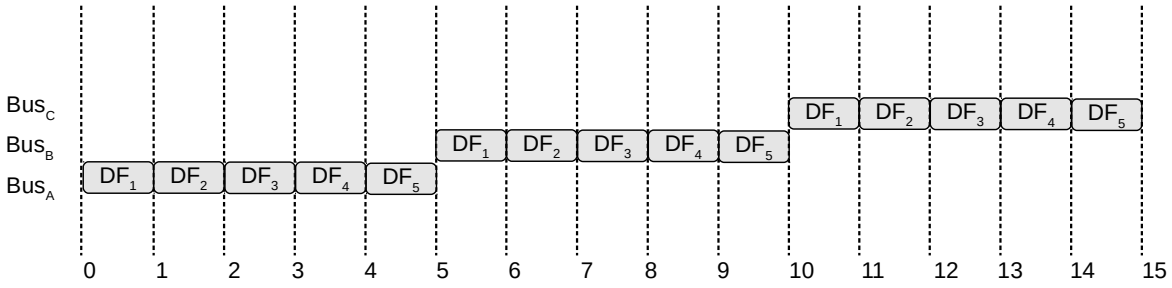
The third scenario (Figure 3.2d) involves two distinct PEs from two distinct PUs for the producer and the consumer. Therefore, data need to be transferred from the producer's PU local memory to the consumer's PU local memory according to the following steps, marked with respectively numbered arrows in Figure 3.2d:

- *Step* (1): The producer writes data in its PUs local memory
- *Step* (2): The DMA of the producer's PU reads data written in *Step* (1)
- *Step* (3): Same DMA writes data to the consumer's PU local memory
- *Step* (4): The consumer reads data written in *Step* (3)

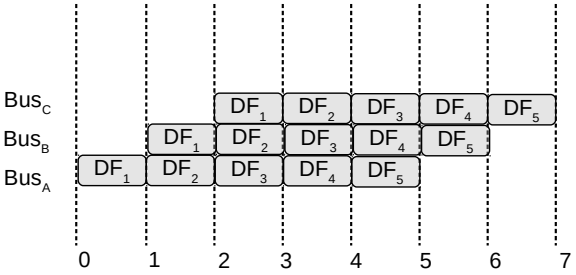
Step (1), *Step* (2) and *Step* (4) are accomplished in negligible time according to *Assumption* 2.

Step (3) requires an inter-PU exchange of data. We call this exchange *a data transfer*. A data transfer is mapped to a route of n buses and $n-1$ bridges.

Remark. In the case where two PEs need to communicate via an external shared memory, the communication via the shared memory can be captured, using our modeling, as follows: a virtual task can be introduced between the two communicating tasks, such that this task has an execution time which is equal to zero. This task is allocated to the shared memory and is in charge of 1) receiving data from the PE hosting the producer task, and 2) sending it to the PE hosting the consumer task. Thus, the two data transfers to/from the shared memory can be mapped as in *Step* (3).



(a) A non-pipelined data transfer. 15 time slots are needed to complete the data transfer.



(b) A pipelined data transfer. Only 7 time slots are needed to complete the data transfer.

Figure 3.3: A data transfer where 5 data fragments ($DF_k, k \in \{1 \dots 5\}$) are transferred on a route of 3 buses, Bus_A, Bus_B and Bus_C in order.

- *Assumption* 6: The preemption of tasks is not allowed, neither their migration at run-time.
- *Assumption* 7: We consider time to be divided in atomic *time slots* that are used as a common time-line to determine the scheduling. As our work is positioned at a high level of abstraction, we simplify the problem by considering that there exists a common time base (governed by means of a *global clock*) that allows the definition of a slot common to all clock domains, in case multiple clock domains are present. If the clocks frequencies are very different, the approach will be limited to only allow a coarse grain analysis, especially for high frequency domains. Assuming a unified global clock is commonly considered in similar work, see for example [75]. In the rest of this manuscript, all reference to time units by “cycle” or “clock cycle” refer to a cycle of the common global clock.
- *Assumption* 8: For a data transfer on a route with $n > 1$ buses, data is trans-

ferred from a bus to another in a *pipelined* fashion. Pipelining aims at sending data through multiple bus segments, in parallel, on a given route. Data is divided into a series of data fragments, and each bus sends a data fragment, as soon as it receives it. As shown in Figure 3.3, pipelining data transfers on buses reduces the data transfer time compared to a non-pipelined transfer. This pipelined transfer mode also assumes that the time to cross a bridge unit, interposed between two buses, is negligible, as data fragments are forwarded directly without buffering.

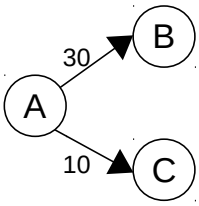
- *Assumption* 9: A data transfer on a route with n bus segments, such that $n > 1$, does not need to allocate at once all buses all along the route and during all the transfer, but only the ones necessary at each slot. Hence, bus segments can be kept idle when not used and thereby, power consumption can be reduced.

- *Assumption* 10: For a given bus, a time slot can be shared by several data transfers, as long as the total amount of data transferred does not exceed the bandwidth of the bus during the time slot.

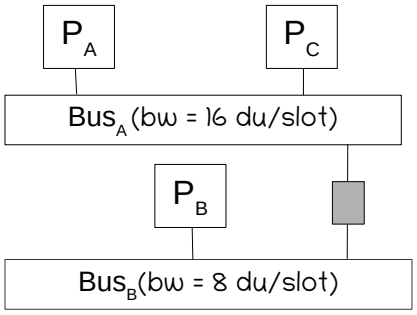
- *Assumption* 11: For a data transfer on a route with n bus segments, such that $n > 1$, we consider that all bus segments operate at the bandwidth of the slowest bus. The residual bandwidth can be used for other data transfers.

Figure 3.4 gives a concrete example of communications mapping and scheduling, as defined by the assumptions discussed above. Here, we assume that tasks A, B and C are mapped respectively to mono-PE PUs P_A, P_B, and P_C, and that communication from A to B requires a data transfer between P_A and P_B, which is routed to {P_A, Bus_A, *bridge*, Bus_B, P_B}. We also assume that communication from A to C requires a data transfer, which is routed to {P_A, Bus_A, P_C}. The grey rectangle in Figure 3.4b represents a *bridge*.

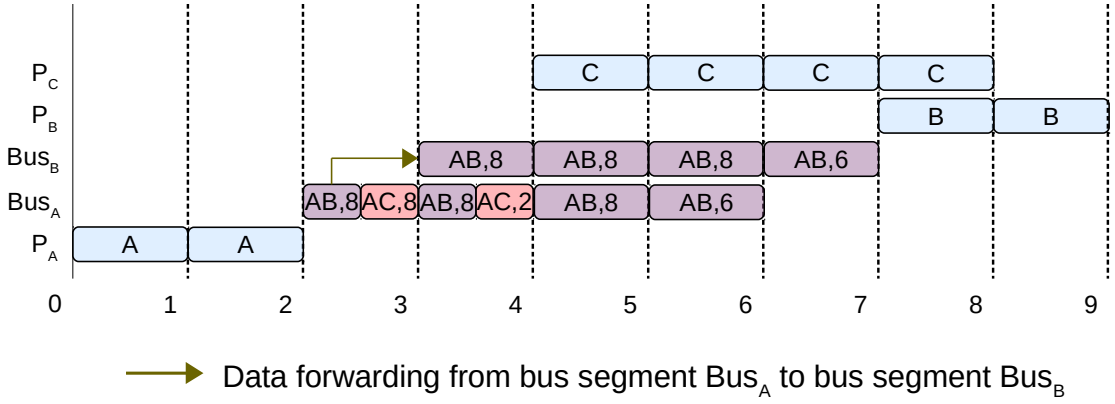
The deployment solution is illustrated in Figure 3.4c using a Gantt chart: Data fragments transferred from A to B at slots 3, 4, 5 and 6 through Bus_A are transferred, in a pipelined fashion, respectively at slots 4, 5, 6 and 7 through Bus_B (*Assumption* 8). Data fragments transferred from A to C cross a single bus route. During slot 7, Bus_A is free. Unless it is used by other applications, it can be kept idle (*Assumption* 9). Bus segment Bus_A is shared between the two data transfer (A to B and A to



(a) An application excerpt



(b) A candidate architecture



(c) The Gantt chart showing a possible mapping and scheduling of the application excerpt (a) to candidate architecture (b).

Figure 3.4: An example showing communications scheduling on a route. A communication on a bus between two tasks X and Y during a slot is denoted XY,N where N is the size of the data fragment in arbitrary data units (*du*).

C) during time slots 3 and 4. Its total bandwidth per slot ($bw = 16\ du/slot$) is not exceeded, as mentioned in *Assumption* 10. The data transfer from P_A to P_B, is performed at the bandwidth of Bus_B, the slowest bus, as indicated by *Assumption* 11. At slots 3 and 4, residual bandwidth on Bus_A is used for communication AC.

3.6 The SMT formulation

As mentioned in the previous Section, our solution is based on splitting time into atomic time slots that are used as a time-line to determine the scheduling. The

length of a single slot, denoted l_{slot}, defines the *granularity* at which the scheduling is analyzed. For example, with $l_{slot} = 1\ cycle$, the system can be analyzed at the precision of a clock cycle. This granularity impacts the precision but also the computational cost of the DSE process. The shorter is l_{slot}, the higher is the precision, but the more expensive is the DSE process. The value of l_{slot} should be selected to determine a desired compromise between the precision and the execution run-time of DSE. The values of variables that are related to timing aspects, in the formulation below, are integer multiples of l_{slot}. The notations for our formulation are described in Table 3.1.

3.6.1 Decision variables

Variable $x_{t,p}$ denotes the mapping decision of task t to a PU p. It is a Boolean variable that takes the value 1 when the task is mapped to the unit and 0 otherwise. Note that, for a given task t, variable $x_{t,p}$ is only defined for the set of compatible PUs, returned by function $F'(t)$. This avoids the creation of variables for mapping options, a priori known unfeasible.

$$\forall t \in \mathcal{T}, \forall p \in F'(t),\ x_{t,p} = \begin{cases} 1 & \text{if task } t \text{ is mapped to PU } p \\ 0 & \text{otherwise} \end{cases} \tag{3.1}$$

Boolean variable $z_{t,c}$ denotes the mapping of a task t to a particular PE from the mapped PU. Figure 3.5 illustrates the difference between variables $x_{t,p}$ and $z_{t,c}$. Similarly to $x_{t,p}$, for a given task t, $z_{t,c}$ is only defined for the set of compatible PEs, returned by function $F(t)$.

$$\forall t \in \mathcal{T}, \forall c \in F(t),\ z_{t,c} = \begin{cases} 1 & \text{if task } t \text{ is mapped to PE } c \\ 0 & \text{otherwise} \end{cases} \tag{3.2}$$

Variable $y^{t_m,t_n}_{i,j,\rho_{i,j}}$ is a boolean variable which indicates the mapping (routing) of a communication c_{t_m,t_n} to a route $\rho_{i,j}$ from the architecture. The set of routes between all pairs of PUs on the architecture are pre-computed. The complexity of generating all routes on the architecture graph is the one of common graph traversal algorithms (e.g., Depth First Search (DFS)). To reduce the cost of exploration, only routes between PUs able to execute producer and consumer tasks are explored. It allows to

Table 3.1: Notations for the SMT formulation

Notation	Definition
$\mathcal{W}$	Set of DAGs of all applications of workload
A	DAG of an application A from workload
$\mathcal{T}$	Set of tasks of all applications of workload
$\mathcal{C}$	Set of communications of all applications of workload
T^A	Set of tasks of applications A
C^A	Set of communications of application A
$deadline^A \in \mathbb{N}$	Deadline of application $A \in \mathcal{W}$ in time slots
$t \in \mathcal{T}$	Task
$c_{t_m,t_n} \in \mathcal{C}$	Communication between tasks t_m and t_n
$bin_t \in \mathbb{N}$	Total amount of input data consumed by one execution of task t in arbitrary data units (*du*)
$bout_t \in \mathbb{N}$	Total amount of output data produced by one execution of task t in arbitrary data units (*du*)
$d_{t_m,t_n} \in \mathbb{N}$	Amount of data to transfer on c_{t_m,t_n} in arbitrary data units (*du*)
$P \subset U$	Set of PUs in target architecture
$C \subset U$	Set of PEs in target architecture
$B \subset U$	Set of buses in target architecture
$c \in C$	PE in target architecture
$p \in P$	PU in target architecture
$b \in B$	Bus in target architecture
$mem_c \in \mathbb{N}$	Amount of available space in local memory of c in arbitrary data units(*du*)
R	Set of routes in target architecture between all couples of PUs
$\rho_{i,j} \in R$	Any route in architecture between PUs i and j
$R_{b1,b2} \subset R$	Set of routes including the bus-to-bridge-to-bus $\{b1, b2\}$ sequence
$bw_b \in \mathbb{N}$	Bus b bandwidth per atomic time slot
$et(t,c) : \mathcal{T} \times C \to \mathbb{N}$	Function returning execution time of task t on PE c in time slots
$F(t) : \mathcal{T} \to C$	Function returning the set of PEs c where task t can execute
$F'(t) : \mathcal{T} \to P$	Function returning the set of PUs p where task t can execute
$deadline^{\mathcal{W}} \in \mathbb{N}$	Upper bound of number of time slots allocated to any unit in target architecture, for any application $A \in \mathcal{W}$, defined as $deadline^{\mathcal{W}} = \max\{deadline_A, A \in \mathcal{W}\}$
$s \in \{0, \ldots, deadline^{\mathcal{W}}\}$	Time slot index
$l_{slot} \in \mathbb{N}$	Length of a single time slot in clock cycles

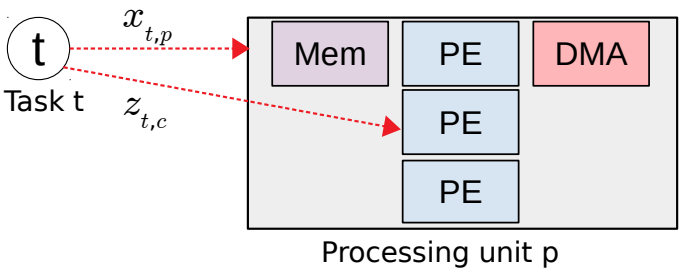


Figure 3.5: The mapping of a task t to a PU p and in particular to a PE $c \in p$, denoted by decision variables $x_{t,p}$ and $z_{t,c}$ respectively.

explore multiple routes between a pair of PUs i and j where tasks t_m and t_n can be mapped.

$$\forall c_{t_m,t_n} \in \mathcal{C}, \forall i,j \in F'(t_m) \times F'(t_n), \forall \rho_{i,j} \in R, y^{t_m,t_n}_{i,j,\rho_{i,j}} = \begin{cases} 1 & \text{if } c_{t_m,t_n} \text{ is routed to route} \rho_{i,j} \\ & \text{between PUs } i \text{ and } j \\ 0 & \text{otherwise} \end{cases} \tag{3.3}$$

$start_t$ is the starting time slot of task t. If a deadline is defined for the application, it can be used as an upper bound on $start_t$ (Equation (3.4)). Otherwise, $start_t$ is defined on $\mathbb{N}$.

$$\forall t \in T^A, start_t \in \{0, \ldots, deadline^A\} \tag{3.4}$$

Variable $e^{t_m,t_n}_{b,s}$ is a positive integer which denotes the amount of data transferred through bus b, at time slot s, for communication from task t_m to task t_n. The domain on which slots are defined can be upper bounded by $deadline^{\mathcal{W}}$. Otherwise, it corresponds to $\mathbb{N}$. $e^{t_m,t_n}_{b,s}$ cannot exceed the bandwidth of the bus bw_b per time slot.

$$\forall c_{t_m,t_n} \in \mathcal{C},\ \forall b \in B,\ \forall s \in \{0, \ldots, deadline^{\mathcal{W}}\},\ e^{t_m,t_n}_{b,s} \in \{0, \ldots, bw_b\} \tag{3.5}$$

3.6.2 Constraints

Task mapping constraints

Constraints (3.6), (3.7) and (3.8) ensure a valid spatial mapping of a task t. In (3.6), a unique and feasible task-to-PE mapping is granted. Constraint (3.7) indicates that task t is assigned to the PU p including PE c. Constraint (3.8) states that a task can be mapped on a PE only if the latter provides enough memory to accommodate its input and output data.

$$\forall t \in \mathcal{T}, \sum_{c \in F(t)} z_{t,c} = 1 \tag{3.6}$$

$$\forall t \in \mathcal{T}, \forall p \in P, x_{t,p} = \begin{cases} 1 & \text{if } \sum\limits_{c \in p} z_{t,c} \geq 1 \\ 0 & \text{otherwise} \end{cases} \tag{3.7}$$

$$\forall t \in \mathcal{T},\ \forall c \in F(t),\ (bin_t + bout_t) \times z_{t,c} \leq mem_c \tag{3.8}$$

Communication mapping constraints

Between a pair of PUs i and j, one or several routes are available. Constraint (3.9) ensures a unique and valid route is selected for communication c_{t_m,t_n} when tasks t_m and t_n are mapped respectively to PUs i and j.

$$\begin{aligned} &\forall c_{t_m,t_n} \in \mathcal{C},\ \forall i,j \in F'(t_m) \times F'(t_n), i \neq j, \\ &\sum_{\rho_{ij} \in R} y_{i,j\rho_{ij}}^{t_m,t_n} = \begin{cases} 1 & \text{if } x_{t_m,i} = 1 \wedge x_{t_n,j} = 1 \\ 0 & \text{otherwise} \end{cases} \end{aligned} \tag{3.9}$$

We highlight that functions $F(t)$ and $F'(t)$ return respectively the sets of PEs or PUs where a task t can be mapped. This differentiation allows our formulation to reduce the number of variables that are allocated to study communications to/from t. In fact, a PE models either a stand-alone single-core unit (e.g., a single-core CPU, a hardware accelerator) or a unit within a multi-core unit (e.g., one core in a multi-core CPU or a DSP). In this second case, variables for communications must be allocated to consider the reception and dispatch of data at the level of abstraction of the entire multi-core unit rather than for each core. We recall that we assume negligible intra-PU communications overhead, *Assumption* 2, Section 3.4.1.

PE capacity constraints

Constraint (3.10) ensures that a PE executes a single task at a time (*Assumption* 1, Section 3.4.1). Therefore, on a given PE, the execution intervals of any two different tasks cannot overlap.

$$\begin{aligned} &\forall t, t' \in \mathcal{T}, t \neq t',\ \forall c \in F(t) \cap F(t'), \\ &\neg((z_{t,c} = 1) \wedge (z_{t'c} = 1)) \vee ((start_{t'} > end_t) \vee (start_t > end_{t'})) \end{aligned} \tag{3.10}$$

The time slot index end_t denotes the end of execution of task t. It is calculated using expression (3.11).

$$\forall t \in \mathcal{T},\ end_t = start_t + \sum_{c \in F(t)} et(t,c) \times z_{t,c} - 1 \tag{3.11}$$

Precedence constraints

Constraint (3.12) guarantees that the scheduling respects the precedence relations defined by the application graph. A consumer cannot start executing before the producer has entirely completed.

$$\forall c_{t_m,t_n} \in \mathcal{C},\ start_{t_n} > end_{t_m} \tag{3.12}$$

Another precedence constraint must also be enforced between tasks and communications, to respect data dependencies: 1) a communication cannot start before the producer task has completed execution, and 2) a consumer task can start as soon as the communication is completed. Given a communication c_{t_m,t_n}, constraints (3.13) allocate no bus bandwidth in time slots that precede the end of producer task t_m or follow the start of consumer task t_n.

$$\begin{aligned} &\forall c_{t_m,t_n} \in \mathcal{C},\ \forall b \in B,\ \forall s \in \{0, \ldots, deadline^{\mathcal{W}}\}, \\ &((s > end_{t_m}) \vee (e_{b,s}^{t_m,t_n} = 0)) \wedge ((s < start_{t_n}) \vee (e_{b,s}^{t_m,t_n} = 0)) \end{aligned} \tag{3.13}$$

Scheduling on buses constraints

The remaining constraints describe the time slots allocation on buses. When a bus b is selected to explore the possibility of being allocated for a data transfer, the bandwidth per time slot of b, bw_b, can be allocated entirely to a data transfer or shared

between different data transfers. Because the buses that form a route $\rho_{i,j}$ can have different bandwidths, our modeling assumptions impose that all buses on a route operate at the bandwidth of the slowest bus. This avoids buffer overflows in bridges and is consistent with our assumption that data are not lost in the interconnect. The difference between the bandwidth of the bus and the bandwidth of the slowest bus on the route can be allocated to other transfers, *Assumption* 11, Section 3.5.

Constraints (3.14) ensure for a bus to respect its bandwidth, on each time slot:

$$\forall b \in B,\ \forall s \in \{0, \ldots, deadline^{\mathcal{W}}\},\ \sum_{c_{t_m,t_n} \in \mathcal{C}} e_{b,s}^{t_m,t_n} \leq bw_b \tag{3.14}$$

Constraint (3.15) ensures for a given communication c_{t_m,t_n} that each bus b which is part of the route $\rho_{i,j}$ where the communication c_{t_m,t_n} is mapped, is assigned the exact amount of data d_{t_m,t_n} of communication c_{t_m,t_n}. For a given communication, this constraint works together with Constraint (3.9), which enforces that the communication is mapped to at most one route. Thus, *all buses* of *the possibly mapped route*, transfer *all data* of the communication.

$$\begin{aligned} &\forall b \in B,\ \forall c_{t_m,t_n} \in \mathcal{C}, \\ &\sum_{s \in \{0,\ldots,deadline^{\mathcal{W}}\}} e_{b,s}^{t_m,t_n} = d_{t_m,t_n} \times \sum_{\substack{i,j \in F'(t_m) \times F'(t_n) \\ \rho_{i,j} \in R, b \in \rho_{i,j}}} y_{i,j,\rho_{i,j}}^{t_m,t_n} \end{aligned} \tag{3.15}$$

Constraint (3.16) corresponds to the pipelining of data between two bus segments as described in *Assumption* 8, Section 3.5. Data transferred on bus b_1 at time slot s are transferred on next bus b_2 at time slot $s+1$. Note that we use the subset of routes $R_{b1,b2}$ defined in Table 3.1.

$$\begin{aligned} &\forall c_{t_m,t_n} \in \mathcal{C},\ \forall s \in \{0, \ldots, deadline^{\mathcal{W}}\},\ \forall b_1, b_2 \in B \\ &\sum_{\rho_{i,j} \in R_{b1,b2}} y_{i,j,\rho_{i,j}}^{t_m,t_n} = 1 \implies e_{b_2,s+1}^{t_m,t_n} = e_{b_1,s}^{t_m,t_n} \end{aligned} \tag{3.16}$$

3.6.3 Latency design objectives

For a workload with one or multiple applications, latency of an application A, denoted $latency_A$, is defined as the time that elapses between the start of execution of the workload and the end of execution of application A. For each application of the studied workload, $A \in \mathcal{W}$, a design objective on latency can be considered to

guarantee that a deadline is respected (Constraint (3.17)), or to find the minimum latency (Objective function (3.18)).

$$latency_A \leq deadline_A \tag{3.17}$$

$$Minimize\ latency_A \tag{3.18}$$

When a workload is composed of multiple applications, to satisfy deadline design objectives of multiple applications, Constraint (3.17) can be simply applied to one, several, or all application(s) of the workload. This is expressed by Constraints (3.19), such that $\mathcal{W}_D \subset \mathcal{W}$ is the subset of applications from $\mathcal{W}$ for which deadlines should be respected.

$$\forall\ A \in \mathcal{W}_D,\ latency_A \leq deadline_A \tag{3.19}$$

To minimize latency of multiple applications from the same workload, the formulation can be used in different ways to achieve different design requirements. We propose the two following options.

- Minimize latency of the *overall workload*, *that is minimize the time that elapses between the start of execution of the workload, i.e., all applications, and its end of execution*. This can be achieved by introducing a new variable $latency_{\mathcal{W}}$ that denotes latency of the overall workload, and applying Constraint (3.20) and Objective function (3.21).

$$\forall\ A \in \mathcal{W},\ latency_{\mathcal{W}} \geq latency_A \tag{3.20}$$

$$Minimize\ latency_{\mathcal{W}} \tag{3.21}$$

- If rather than the latency of the overall workload, the latency of each single application is of interest, the sum of latencies can be aggregated in a weighted objective function such that different (or equal) priorities are given to applications. This is captured in Objective function (3.22), such that each application $A \in \mathcal{W}$ is associated with a weight w_A denoting its priority. The latter can be equal to zero for a given application A_k to indicate that no minimization of latency is required for application A_k.

$$Minimize \sum_{A \in \mathcal{W}} w_A \times latency_A \tag{3.22}$$

Remark. For the same workload, different design objectives on latencies of each single application can be combined. For example, one can be interested in satisfying deadlines for all applications of the workload but one for which latency is minimized.

3.7 Implementation

In order to assess the applicability and usability of the proposed DSE approach, we integrated it to an existing Model-Driven Engineering (MDE) tool for the design of embedded systems. This Section first details how input models (applications and architectures) can be captured inside a MDE tool, and second, explains how the SMT formulation proposed in previous Section can be generated and solved using a state-of-the-art solver.

3.7.1 Overview of the implemented solution

TTool [2, 3] is a free and open-source UML/SysML tool that includes several UML profiles, including the DIPLODOCUS environment that targets hardware/software partitioning. TTool/DIPLODOCUS supports several design stages from capturing application and architecture models using UML/SysML diagrams to code generation.

We selected TTool/DIPLODOCUS to integrate our DSE approach, as it is free, open-source and targets the system-level design of embedded systems, taking into account both hardware and software aspects of these systems. An overview of the implemented DSE workflow is illustrated in Figure 3.6. Application and architecture models are first created by means of TTool/DIPLODOCUS diagrams. In Figure 3.6, UML/SysML diagrams are parsed and analyzed in order to automatically generate a SMT formulation. The latter is given as input to the state-of-art SMT solver Z3 [19] which returns a solution if the formulas are satisfiable. The output model is converted back to UML/SysML: the initial diagrams are annotated with the mapping solution. Gantt charts are created to illustrate the mapping and scheduling of tasks and communications. TTool/DIPLODOCUS and our DSE extension are written in Java.

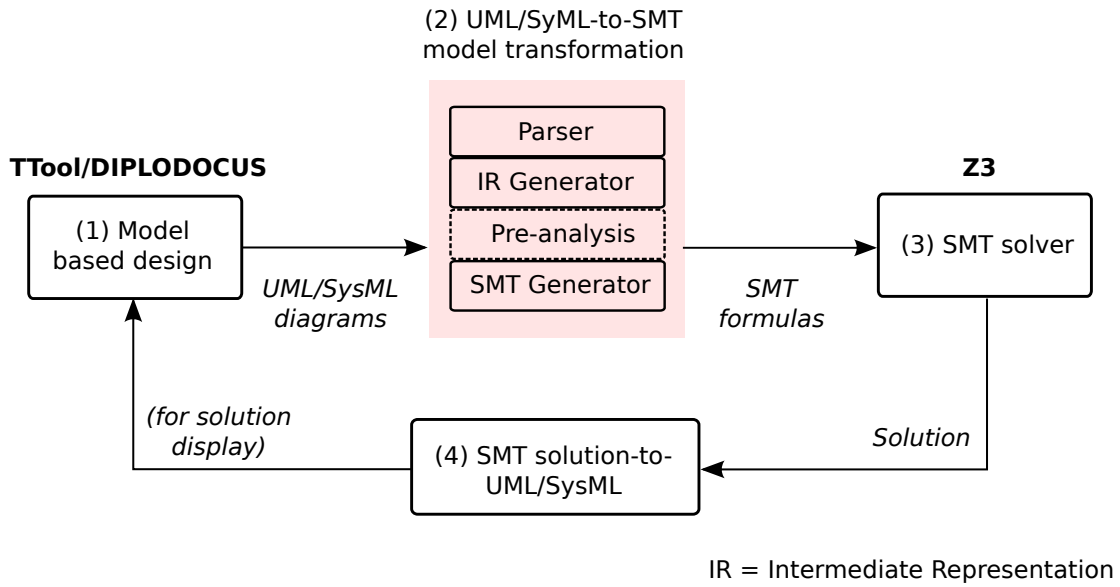


Figure 3.6: Overview of the design space exploration workflow. The explanation of the pre-analysis sub-module will be given in Chapter 4.

3.7.2 Model-based design

Workload model

In TTool/DIPLODOCUS, each application model from the studied workload is captured with SysML Block Diagrams and UML Activity Diagrams. An application model can be described as a set of blocks interconnected by data dependencies. Blocks refer to tasks in the scope of this work. Figure 3.7 shows an example block diagram of an application model in in TTool/DIPLODOCUS.

A shown in Figure 3.8, the block diagram captures for each task the operation type and, specifically for the sink task, the application deadline, as explained in Section 3.3. Activity diagrams describe for each block (task) its behavior as the steps required for its execution, i.e., processing steps, memory access steps:

- A data processing step is described in TTool/DIPLODOCUS by its *algorithmic complexity*. The algorithmic complexity (denoted EXECI) is an abstract complexity measure that denotes the number of processing operations that are

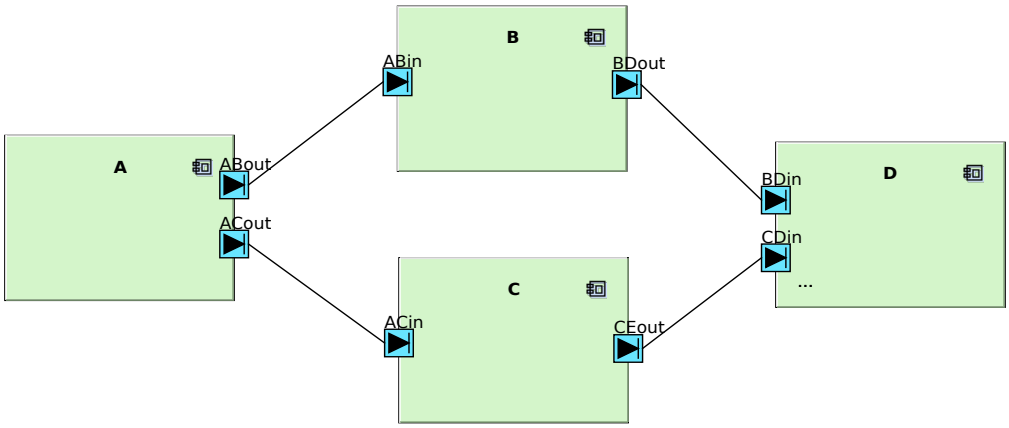


Figure 3.7: A SysML Block Diagram of an application model in TTool/DIPLODOCUS. Blocks represent tasks while links between blocks represent data dependencies.

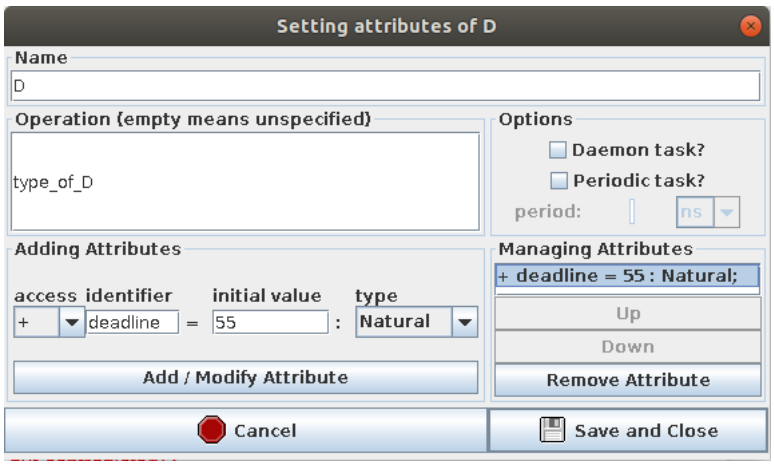


Figure 3.8: Parameters of task D from Figure 3.7.

required to complete a task (e.g., number of integer operations). A user can either rely on this representation or directly provide execution times of tasks on PEs.

- A memory access step defines the number of *data packets* to read/write by the task. The amount of data d_{t_m,t_n} of a data dependency, defined in Section 3.3, corresponds in TTool/DIPLODOCUS to the product of the number of data packets and the size of a single data packet in data units (*du*).

As an example, activity diagram in Figure 3.9 describes the steps required for the execution of task B from Figure 3.7. It shows that task B first reads 10 data packets

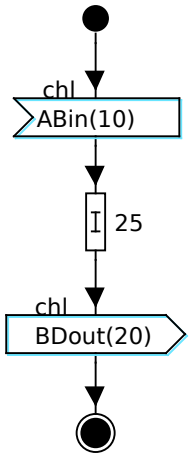


Figure 3.9: UML activity diagram of task B from Figure 3.7.

sent by task A, then performs a processing step which EXECI is equal to 25, and finally produces 20 data packets in destination to task D.

Architecture model

An architecture is captured with a UML Deployment diagram where nodes are used to model architecture resources while links between nodes identify the physical links between the architecture resources. A resource can be a PU, a bus or a bridge. Since we assume that memories are integrated into PUs, as explained in Section 3.4, we don't model physical memories separately. Figure 3.10 shows an example of a UML deployment diagram for an architecture with 4 PUs, 2 buses and 1 bridge.

Buses are characterized by their bandwidth. Each PU block captures the set of supported operations, the performance metric, the number of PEs encapsulated in the unit and the amount of available memory. The *performance metric* denotes the number of clock cycles taken by the unit to execute a single processing operation (e.g., an operation on integer values). If not provided directly, the execution time of a task on a PE is computed from the algorithmic complexity (EXECI) expressed in the UML Activity diagram of a task (e.g., the 25 parameter in Figure 3.9) and the performance metric.

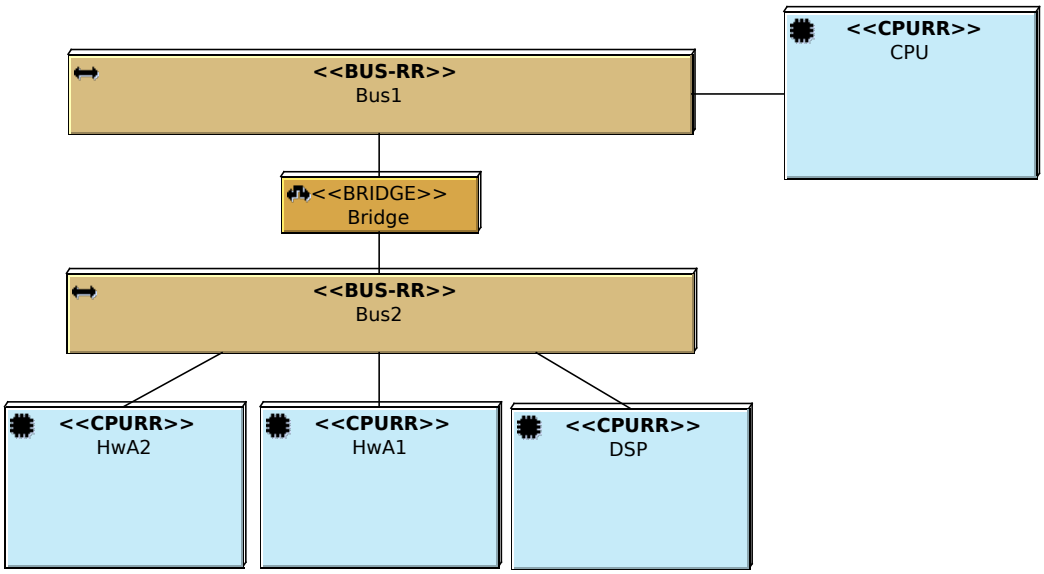


Figure 3.10: A UML Deployment diagram of an architecture model in TTool/DIPLODOCUS.

3.7.3 UML/SysML-to-SMT transformation

The solver operates on a formulation in the form of variables and constraints expressed in first-order logic. Therefore, input models and design constraints need to be translated to a first-order logic formulation. We implemented algorithms needed to analyze input models, perform this transformation and feed the solver with the formulation. The following modules, illustrated in Figure 3.6, are implemented.

- **Parser:** It parses the software objects representing the input UML/SysML diagrams that define the workload and the target architecture models.

- **IR Generator:** This sub-module represents a key part of our DSE engine. Relying on the model-based diagrams, it creates the intermediate representation (graphs) of input models as defined in Sections 3.3 and 3.4. It performs analysis in order to generate an input instance which contains all information needed for the creation of the SMT model. For example, it identifies for each task the set of compatible PUs, calculates execution times on these units, generates the paths (routes) that are available in the architecture graph between each pair of PUs, etc.

- **SMT Generator** This sub-module is responsible for the conversion of input models and design constraints to SMT variables and constraints which will be provided as input to the SMT solver. We integrated the Z3 SMT solver tightly with the DSE engine and TTool/DIPLODOCUS.

3.7.4 Model-based deployment solution

We also implemented algorithms that retrieve the solver's output and transform it back to models that describe the deployment solution. In TTool/DIPLODOCUS, UML Deployment diagrams of architecture models are annotated with task mapping decisions: each PU is annotated with the tasks mapped to it. Moreover, Gantt charts are created to illustrate the mapping and scheduling of tasks and communications on processing and communication resources.

3.8 Evaluation

In this Section we evaluate our contribution for the DSE of segmented bus architectures. We consider a set of four real applications taken from [50]: a *Sobel* filter, a *SUSAN* filter, a *RASTA-PLP* application, and a *JPEG* encoder, Figure 3.11.

We will first present a simple case study in sub-section 3.8.1, where we show how the proposed approach can be used to guide the design of a new architecture, by comparing the found optimal solutions for various architectures and selecting the architecture that best meets the latency design objective of a given workload. For this case study, the workload is composed of 4 instances of *SUSAN* running in parallel. Next, in sub-section 3.8.2, we will use all applications of Figure 3.11 to assess the scalability of our approach.

3.8.1 Experiment 1: Best interconnect selection

In this experiment, we will show how the proposed approach can compare optimal solutions, in terms of latency, of different architectures for a given workload. Here, the latency design objective is to minimize latency for each single application, relying on Objective function (3.22), sub-section 3.6.3, such that equal weights of 1 are given to each *SUSAN* instance.

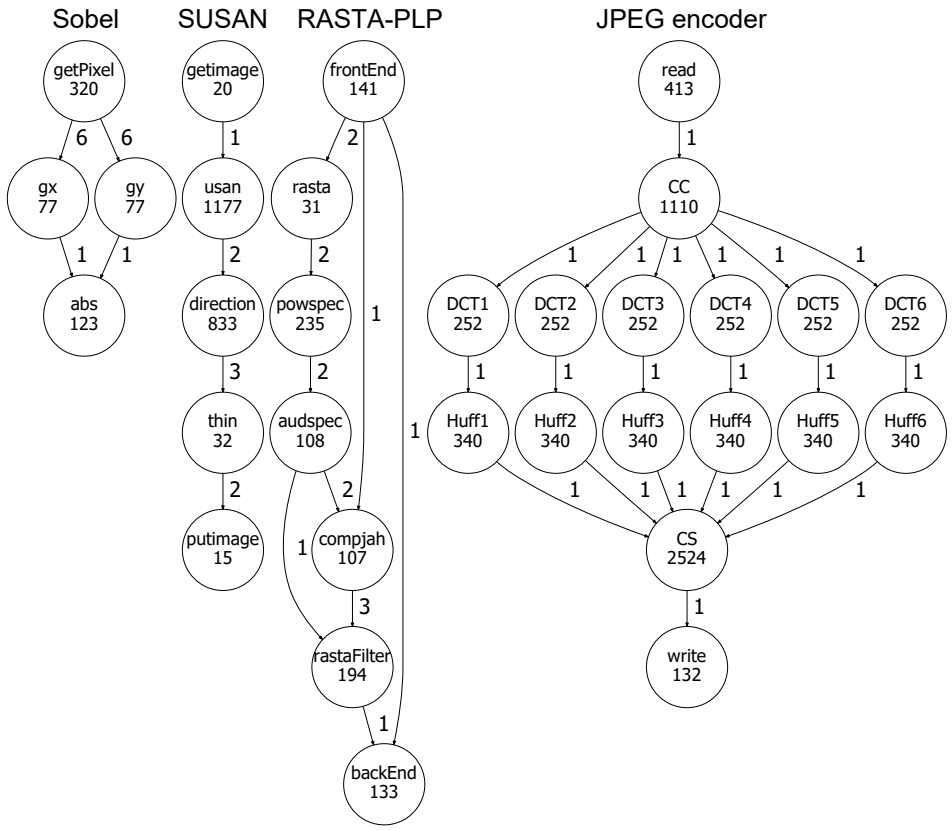


Figure 3.11: The dependency graphs of our testbench. Edges are labeled with the number of exchanged data packets. Tasks are labeled with their execution times.

In all tests, time units for latency and execution times are expressed in time slots and data units (*du*) are used to express communication volumes. In Figure 3.11, tasks are directly annotated with execution times reported in [50] in cycles. In this experiment, the length of a single time slot is set to 1 clock cycle. To account for architectures with heterogeneous processing resources, we assume that execution times are applicable only to generic processing resources. When tasks are allocated to DSPs or hardware accelerators, their execution times correspond respectively to 1/5 and 1/50 of the times reported.

The aim of this experiment is summarized as follows: *Find the best interconnect, among a set of candidates, for a given workload and a given set of processing resources, such that latencies of workload applications are minimized.*

The studied workload consists of 4 instances of application *SUSAN* from Fig-

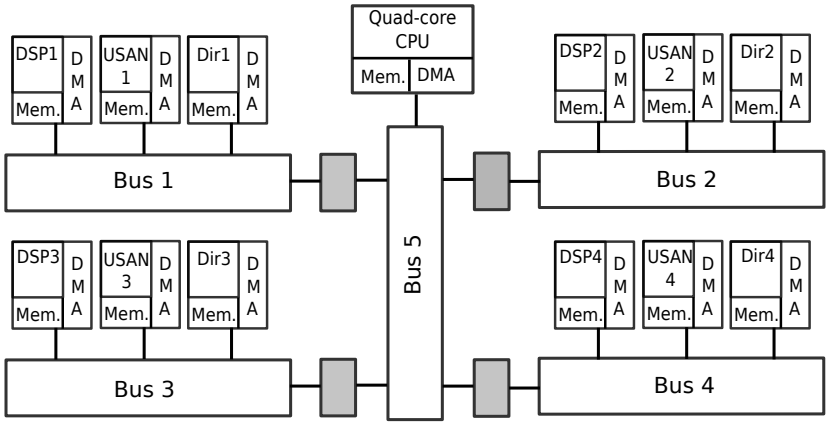


Figure 3.12: Common block diagram of *Architecture 2* and *Architecture 3*.

ure 3.11 where the amount of data d_{t_m,t_n} on each data dependency is equal to the product of the number of data packets (annotated in the Figure) and the size of a single data packet in data units, set to 32 *du*. A set of 13 processing units has been selected beforehand for the target architecture. It consists of: a quad-core CPU and 4 clusters of PUs. Each cluster contains 1 DSP (for task *thin*) and 2 hardware accelerators (for task *usan* and task *direction*). Except for the CPU which contains 4 PEs operating in parallel, all other PUs have each a single PE, capable of executing a single task at once.

We consider 3 interconnect candidates: a first interconnect candidate in an architecture denoted *Architecture 1*, such that all 13 PUs are connected to *a shared bus*, which bandwidth is equal to 16 *du*/slot. The second interconnect candidate (*Architecture 2*), Figure 3.12, is a segmented bus where only the quad-core CPU is connected to a central shared bus that interfaces to 4 segments, each with a cluster of 1 DSP and 2 hardware accelerators. The shared bus and all bus segments in *Architecture 2* have a bandwidth equal to 16 *du*/slot. The third interconnect candidate (*Architecture 3*) has exactly the same topology of the second interconnect, but the central shared has a bandwidth equal to 64 *du*/slot.

We look for the optimal solution that minimizes latency for all applications of the workload, for each architecture candidate.

Gantt charts showing mapping and scheduling solutions for each architecture are illustrated in Figures 3.13, 3.14 and 3.15. Different colors are used to distinguish

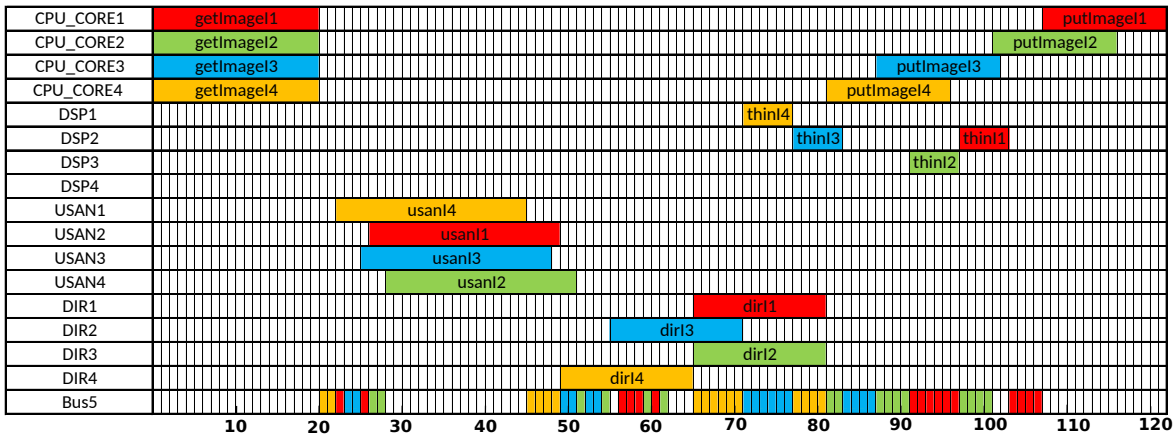


Figure 3.13: Gantt chart for the solution found for *Architecture 1* (1 cell = 1 slot).

tasks and data transfers associated to different *SUSAN* instances. In Figure 3.13, results are given for the shared bus architecture, i.e., *Architecture 1*. First it can be seen that, although all *SUSAN* instances start execution at the same time slot, they finish executing respectively at time slots 96, 102, 116 and 122. If we analyze thoroughly the scheduling, we can see that instances which are delayed (colored in blue, green, and red in Figure 3.13) had to wait at some points of the scheduling for the shared bus to be released by faster instances. For example, the data transfer occurring between *dirI2* and *thinI2* (green) was interrupted as soon as task thinI3 (blue) has finished, and had to wait for the data transfer between *thinI3* and *putImageI3* (blue) to take place before resuming. As a result to these contentions on the shared bus, parallel processing resources are not used efficiently in *Arcihtecture 1*. We can see that cumulative delay led to a completely sequential execution of tasks *thin* of each *SUSAN* instance despite the allocation of parallel processing resources. Note that there are more than one solution optimizing latency. For instance, here, tasks *thin* could have been mapped to the same DSP without changing the values of latencies.

For *Architecture 2*, in Figure 3.14, we first can notice smaller gaps between optimal latencies with respect to *Arcihtecture 1* (98, 102, 106 and 110 slots). This was enabled by distributing processing resources over multiple bus segments, thus enabling more parallel data transfers. Here, only data transfers relating tasks *getImage*

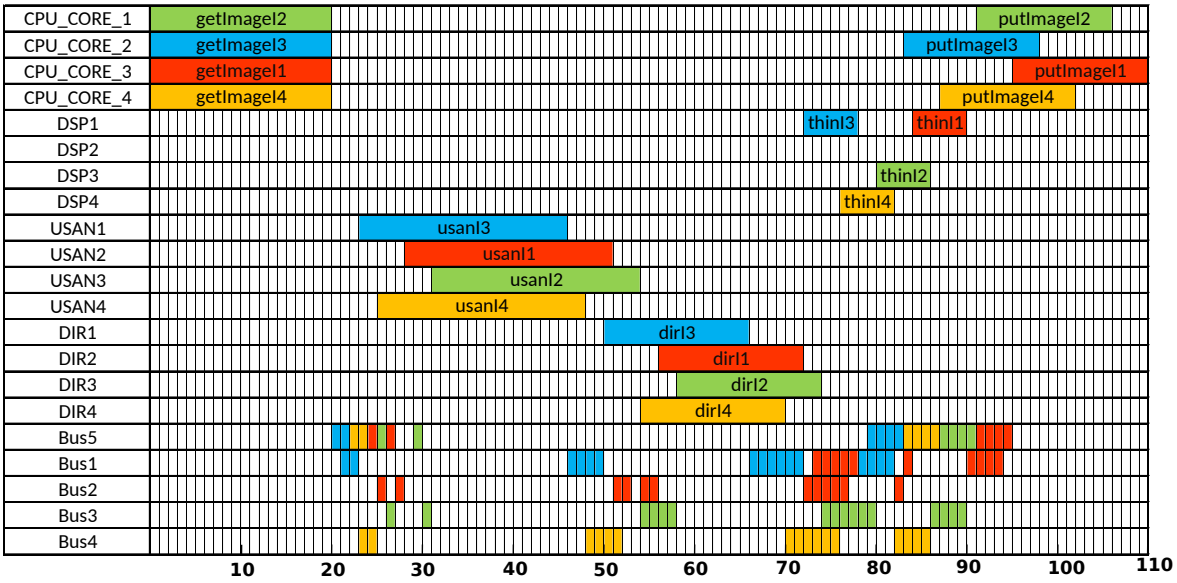


Figure 3.14: Gantt chart for the solution found for *Architecture 2* (1 cell = 1 slot).

to tasks *usan*, and tasks *thin* to tasks *putImage* compete over the shared bus segment (Bus 5). Other data transfers can take place in parallel on local bus segments. However, contentions still exist and prevent from an efficient usage of parallel processing resources. For example, data transfer between *thinI4* and *putImageI4* (yellow) had to wait for Bus 5 to finish the transfer between *thinI3* and *putImageI3* (blue). As a result, task *dirI4* was delayed by 2 slots while it could have started earlier. Similar results are observed for transfers between *thinI2* and *putImageI2* (green), and *thinI1* and *putImageI1* (red).

The pipelining on bus segments can be seen in this scheduling at many points. For instance, on data transfer between *getImageI3* and *usanI3*, we can see that data is first sent on Bus 5 (slots 21 and 22), then on Bus 1 (slots 22 and 23), where the PU hosting task *usanI3* is connected (hardware accelerator *USAN1*). Here, two time slots are allocated on each bus because the capacity of each bus is equal to 16 *du*/slot, and 32 *du*[2] should be sent between *getImageI3* and *usanI3*.

Since in the second interconnect candidate, contentions still occur on the central shared bus resulting into an underutilization of potential parallelism of computations,

[2]We remind that edges in Figure 3.11 are annotated with the number of exchanged data packets and that the size of a single data packet is equal to 32 *du*.

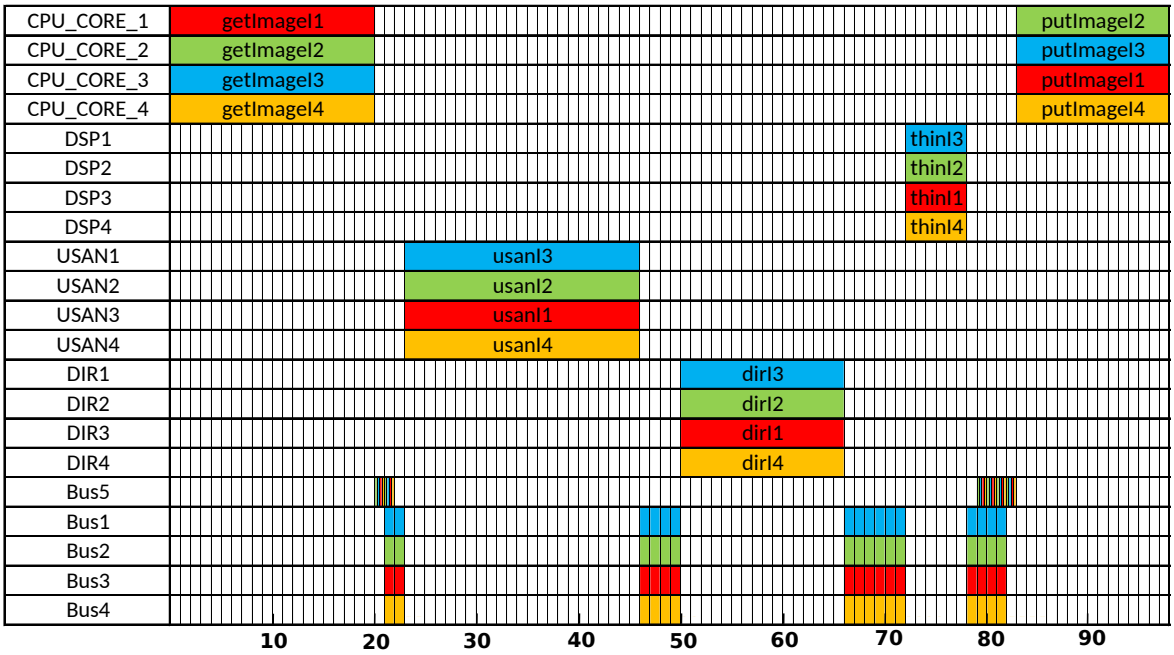


Figure 3.15: Gantt chart for the solution found for *Architecture 3* (1 cell = 1 slot).

we increase the bandwidth of the central shared bus to 64 *du*/slot in *Architecture 3*. The resulting scheduling solution is given in Figure 3.15, where we can see that all *SUSAN* instances start and finish executing at the same time slots. We can also see that computations and data transfers are perfectly parallelized on processing resources and bus segments. For example, as soon as task *getImage* of a each *SUSAN* instance completes executing on a core of the CPU, 16 *du* are transferred on Bus 5 (slot 21) to the respective task *usan*. At the slot level, all *SUSAN* instances perform this transfer in parallel (even if physically the bus performs the transfers sequentially), and the operation repeats on the following slot (slot 22) to transfer the remaining 16 *du* needed for the transfer. Data units which are sent on slots 21 and 22 on Bus 5 are forwarded to buses connecting destination PUs in slots 22 and 23. This parallelism is enabled by the high bandwidth in Bus 5 allowing to transfer 64 *du*/slot. Similar transfer schema can be observed in data transfers between tasks *thin* and tasks *putImage*.

From these results, we can see that the proposed DSE allows to analyze mapping and scheduling on different bus-based architectures for latency optimization. We

showed that the approach allows to analyze contentions on the interconnect due to the presence of multiple applications. In this case study, our DSE framework shows that a segmented interconnect, where segments have a bandwidth of 16 *du*/slot and the shared bus a bandwidth of 64 *du*/slot is the best architecture candidate to the studied workload.

3.8.2 Experiment 2: Scalability evaluation

In this sub-section, we study the scalability of our DSE engine by evaluating the solver run-time evolution as a function of increasing problem sizes. As a first experiment, we evaluate the solver run-time to find a deadline-aware solution (real-time design objective on latency), as defined in Section 3.2. Deadlines were set to 490 cycles (*Sobel*), 1170 cycles (*SUSAN*), 575 cycles (*RASTA-PLP*) and 1830 cycles (*JPEG*). These values correspond to the time requirement to complete one execution for the applications in Figure 3.11. Data dependencies are annotated with the number of produced/consumed data packets (similarly to the graphs defined in [50]). In all experiments of this Section, we assume a data packet has a size of 8 data units, as in [50]. For instance, in application *Sobel*, task *getPixel* sends 48 data units to task *gy*. All Experiments were conducted on a Linux workstation with 64 logical CPUs, clocked at 3.5 GHz and with 64 GB of memory running release 4.8.7 of Z3 SMT solver (default configuration). We consider a set of workloads for all combinations of the applications in Figure 3.11. The target architecture in all experiments of this sub-section is called *Architecture A*, Figure 3.16. It is a MPSoC including 1 quad-core CPU, 9 DSPs and 1 hardware accelerator. The communication architecture is a segmented bus with 3 bus segments. Specialized DSPs and the hardware accelerator are associated with the following operation sets: $O_{DSP1} = \{Huff, DCT\}$, $O_{DSP2} = \{pows, comp, filter\}$, $O_{DSP3} = \{gx, gy\}$, $O_{DSP4} = \{USAN\}$ and $O_{HwA} = \{CS\}$. We set a timeout of 1800 seconds after which we stop the exploration if no solution was found. Since we apply our technique at a high abstraction level, e.g. the abstraction level of TTool / DIPLODOCUS, we expect to perform fast iterations over different architectures: the selected timeout corresponds to a reasonable iteration time for a designer. Moreover, similar timeouts are used for exploration at the same abstraction level [50, 67]. For these reasons,

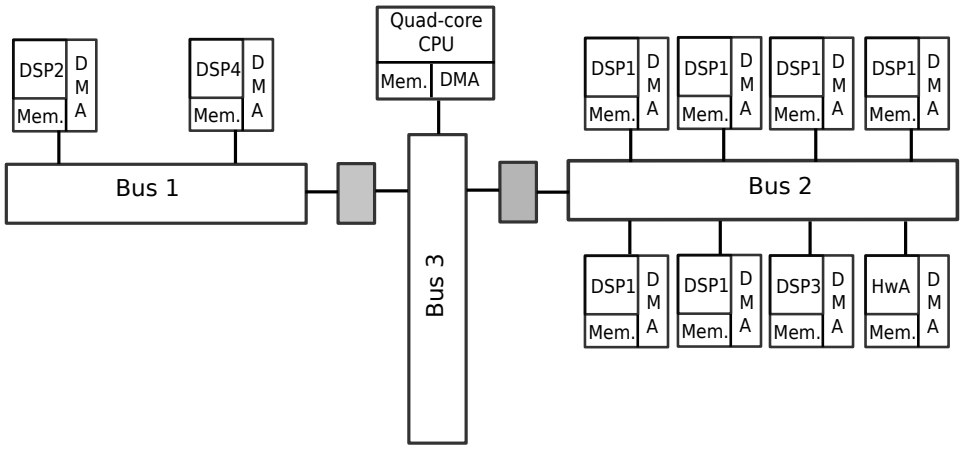


Figure 3.16: Block diagram of *Architecture A*. Specialized DSPs and the hardware accelerator are associated with the following operation sets: $O_{DSP1} = \{Huff, DCT\}$, $O_{DSP2} = \{pows, comp, filter\}$, $O_{DSP3} = \{gx, gy\}$, $O_{DSP4} = \{USAN\}$ and $O_{HwA} = \{CS\}$. Bus 3 has a bandwidth of 64 *du*/slot and Bus 1 and Bus 2 have each a bandwidth of 32 *du*/slot.

the same timeout is considered in all evaluations of this thesis.

Figure 3.17 shows the run-time evolution with a logarithmic scale, as a function of different workloads. A workload, on the horizontal axis refers either to a single application (e.g., *sobel* for the *Sobel filter*, *jpeg* for the *JPEG encoder*), or to a combination of multiple applications (e.g., *sosurajp* for the combination of all four applications). The length of a single slot in this experiment is set to 1 slot = 1 cycle. From the Figure, we can see that the resolution of 7 out of 15 workloads completes within the timeout of 1800 seconds, while the DSE was not able to return a solution for the 8 remaining workloads. For the solved workloads, the run-time to find a deadline-aware solution varies between 3.154 seconds for *sobel* and 1077.377 seconds for *sosura*.

With a granularity of 1 slot = 1 cycle, it is impossible to study 8 out of 15 workloads of our testbench within the timeout. The following experiment aims at studying the impact of granularity and identifying the finest granularity that can be achieved, within the timeout, when solving the heaviest workload (*sosurajp*).

Figure 3.18 shows the solver run-time for workload *sosurajp*, as a function of l_{slot} which varies from 10 cycles down to 1 cycle. It can be seen that, for a granularity ranging from 10 cycles/slot down to 3 cycles/slot, the solver completes within the

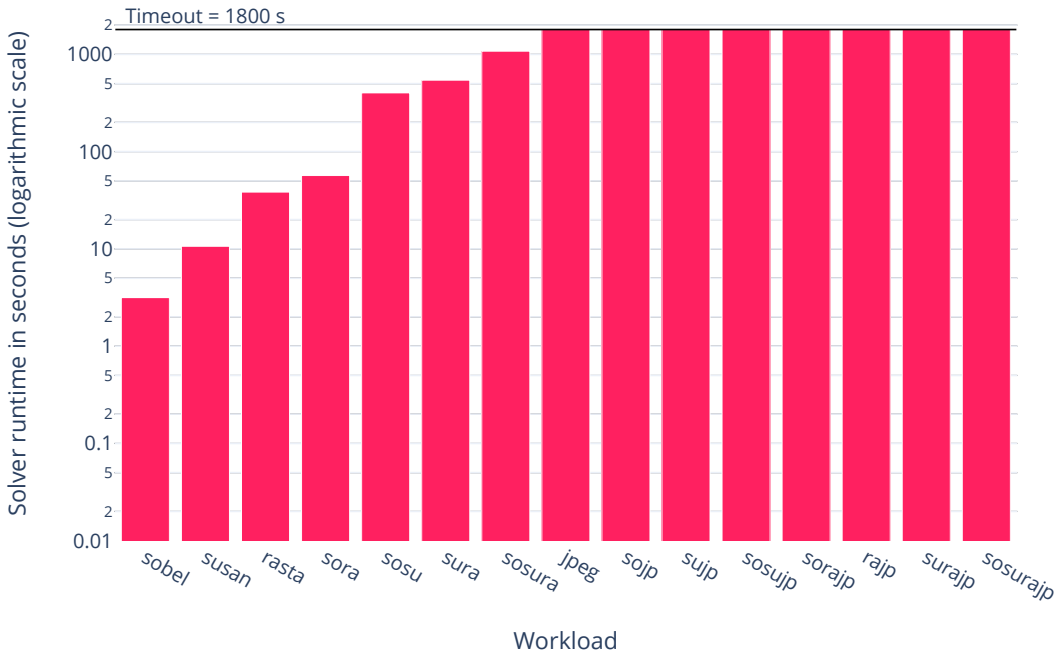


Figure 3.17: Solver run-time to produce a deadline-aware solution, as a function of different workloads, such that granularity is set to 1 slot = 1 cycle.

timeout, while at a finer granularity, the timeout is exceeded. In fact, finer granularity translates to a larger number of time slots, meaning a larger problem, while a coarser granularity reduces the number of time slots and thereby, the size of the problem under study. Thus, larger problems that cannot be explored at a fine grain slot in the allocated time budget, like workload *sosurajp*, can be explored at a coarser grain slot. However, a coarser grain slot may degrade the accuracy due to the abstraction of timing information that take place below the chosen time slot.

3.9 Summary and limitations

This Chapter has presented our SMT formulation to solve the problem of mapping and scheduling both tasks and communications onto architectures with a multi-bus

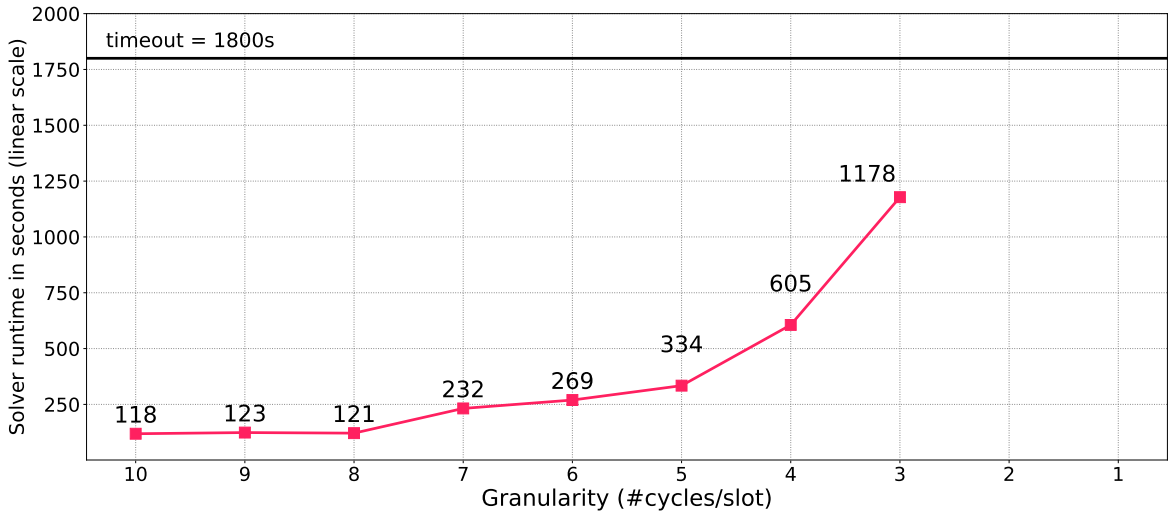


Figure 3.18: Solver run-time to produce a deadline-aware solution, as a function of exploration granularity for workload *sosurajp*.

communication architecture. The proposed modeling accounts for a latency design objective that either minimizes latencies or guarantees real-time deadlines for each application of a studied workload. An experimental analysis using real-world streaming applications demonstrated how our approach can assist a designer to evaluate different communication architecture configurations and to select the configuration that best satisfies the latency design objective.

However, our scalability evaluation (sub-section 3.8.2) showed that, in practice, the proposed approach can only solve small problems within a specified time budget of 30 minutes. In this time budget, only solutions that guarantee real-time deadlines can be practically found, and not for all studied workloads. In fact, the modeling based on time slotted variables allows to analyze contentions and schedule concurrent access to shared resources. However, as shown in the experiment that studied the granularity, Figure 3.18, increasing the number of time slots under study by using a finer grain slot, results into longer exploration run-times and may lead to the impossibility to complete DSE within the specified time budget. In next Chapter, we address this limitation by proposing a technique to reduce the number of time slots under study, thereby, reducing the resolution time and improving scalability.

Chapter 4

A Reduction Method to Prune the Design Space for the Problem of Scheduling Tasks and Communications

4.1 Introduction

In Chapter 3, we presented the first contribution of this thesis which is a SMT formulation for the mapping and scheduling of tasks and communications on architectures with a multi-bus interconnect. The experiments presented in Section 3.8 showed that the time needed for solving the formulation exceeds the allocated time budget (30 minutes) for many workloads of the studied benchmark. Several techniques exist to deal with this complexity such as breaking symmetries that are present in the problem (e.g., symmetric processing resources as in [67]). Unfortunately, in our work, the use of symmetry breaking techniques on the architecture is limited by the heterogeneity of the processing model (e.g., DSPs, CPUs, hardware accelerators) and the irregular topology on the interconnect (i.e., possibly different communication schemes between processing resources). As shown in Figure 3.18, reducing the

number of time slots on which the scheduling is created by using a coarser grain slot improves DSE run-time. Thus, in this Chapter we will investigate how to reduce the number of time slots to speed-up DSE.

In this Chapter, we present our **second contribution**: a new method to prune the design space, prior to problem solving, and accelerate the overall DSE process. The method consists in a *pre-analysis* (step 2 in Figure 4.1) that leverages the problem knowledge in order to identify a subset of non-valid candidates from the design space and eliminate them before the generation of the SMT formulation. Therefore, it doesn't impact the quality, i.e., completeness, of the approach since only non-valid solutions are pruned. The method is not restricted to the scheduling on architectures with a bus-based interconnect but it can intervene for other architectures with different communication architectures as soon as there is a need to analyze different communications scheduling on shared communication resources.

The remainder of this Chapter is organized as follows: For the sake of genericity, Section 4.2 provides a mathematical formulation model in SMT for the problem of scheduling tasks and communications regardless of the communication architecture constraints. It is an excerpt of the formulation proposed in Chapter 3, such that we only keep the variables and constraints which are essential to apply the reduction method. Section 4.3 describes the reduction method and details the algorithms that are used to identify non-valid solutions and prune the design space. Finally, in Section 4.4, we present a set of experiments to evaluate the efficiency of the reduction method (applied on formulation of Chapter 3) to reduce the size of the design space and improve the scalability of the DSE. We also demonstrate how the optimized formulation enabled to find optimal solutions (i.e., lowest-latency design objective) within a timeout for which the initial formulation fails to reach deadline-aware solutions.

4.2 A SMT Model for the Problem of Scheduling Tasks and Communications

The mathematical formalization of a problem first requires modeling decisions into variables. Then, relationships between these variables and other input parameters

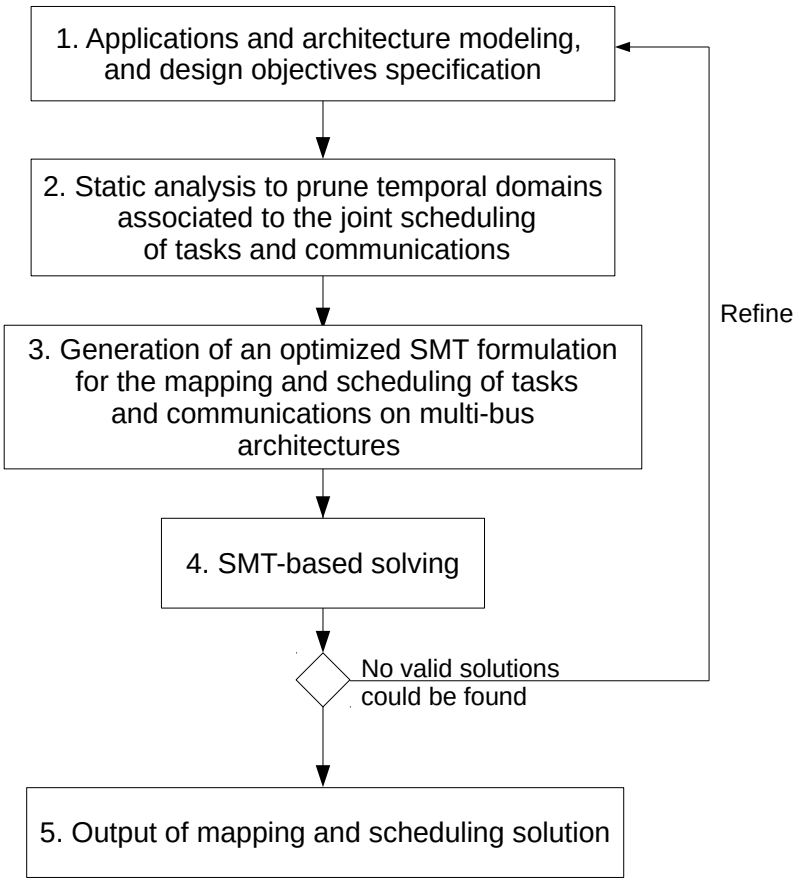


Figure 4.1: An optimized DSE approach for the mapping and scheduling of tasks and communications.

are identified and expressed as constraints. In this Section, we present a generic pattern of variables and constraints for scheduling tasks and communications to which our reduction method can be applied. It is an excerpt of the formulation from Chapter 3, where we exclude architecture-specific constraints (e.g., mapping and scheduling on a segmented bus constraints) to emphasize the versatility of the reduction method. This model can in fact be reused with additional decision variables and constraints that are specific to any other target architecture. For example, constraints on how data is transmitted on communication resources like NoCs can be added to investigate the benefit that the reduction method can bring when studying scheduling on NoC architectures.

4.2.1 Assumptions

In order to apply the method, the scheduling problem has to be compliant with the following assumptions:

- Each application from the studied workload is modeled with a separate dependency graph (e.g., DAG) with a unique source and a unique sink (see Section 3.3 of Chapter 3 for a compliant example).
- For each application $A \in \mathcal{W}$, earliest start time and latest finish time, respectively denoted ES_A and LF_A, are known a priori. The latest finish time can typically correspond to the deadline of the application, or a maximum time budget allocated to the execution of the application. The earliest start time typically corresponds to zero, except for some cases where the application execution has some dependencies to satisfy. For example, to capture dependencies between the execution of different iterations of the same periodic application, the earliest start time of an iteration $m+1$ can be set such that the iteration can only start once some tasks or all tasks of iteration m have completed.
- Execution times on compatible processing resources are known for all tasks.

4.2.2 The SMT Model

The model relies on splitting time into a discrete range of time slots such that each resource from the architecture is allocated, during certain time slots, to tasks and communications. Here, tasks are assumed to execute without preemption. The model reuses the same notations from Table 3.1 in Chapter 3.

The task start time variable The start time variable defines the time slot at which the task starts executing. It is an integer variable defined on the interval $\{ES_A, \ldots, LF_A\}$.

$$\forall A \in \mathcal{W}, \forall t \in T^A, start_t \in \{ES_A, \ldots, LF_A\} \tag{4.1}$$

The slot allocation variable The slot allocation variable defines the amount of data that is transferred on a communication resource b, for a communication c_{t_m,t_n}, on a time slot s. The variable is an integer that is comprised between 0 (i.e., data is not

transferred through the communication resource at the time slot) and the maximum amount of data bw_b that the resource can transfer during a single time slot.

$$\forall A \in \mathcal{W}, \forall c_{t_m,t_n} \in C^A,\ \forall b \in B,\ \forall s \in \{ES_A, \ldots, LF_A\},\ e_{b,s}^{t_m,t_n} \in \{0, \ldots, bw_b\} \quad (4.2)$$

The precedence constraint For dataflow applications, a precedence constraint (same as Constraint (3.12)) should be enforced between tasks that are related by a data dependency: the start time of the receiver task is greater than the end time of the sender task.

$$\forall A \in \mathcal{W}, \forall c_{t_m,t_n} \in C^A,\ start_{t_n} > end_{t_m} \quad (4.3)$$

The data dependency constraint This constraint (same as Constraint (3.13)) allows to schedule on the same time-line tasks and communications by enforcing the order of execution that should be respected in a valid scheduling: first, the sender task completes execution, then, the data transfer is performed, and finally the receiver task starts execution.

$$\begin{aligned} &\forall A \in \mathcal{W}, \forall c_{t_m,t_n} \in C^A,\ \forall b \in B,\ \forall s \in \{ES_A, \ldots, LF_A\}, \\ &\Big((e_{b,s}^{t_m,t_n} = 0) \vee (s > end_{t_m})\Big) \wedge \Big((e_{b,s}^{t_m,t_n} = 0) \vee (s < start_{t_n})\Big) \end{aligned} \quad (4.4)$$

4.3 Description of the Design Space Reduction Method

4.3.1 Overview

In this Section, we present our method to reduce the design space for the problem of scheduling tasks and communications. It applies the problem model introduced in Section 4.2, alone or with additional variables and constraints. The proposed method is structured in two steps that result into two reductions (Figure 4.2). The first step is based on the principle of *constraint propagation*. Constraint propagation includes any reasoning that consists in removing values or combination of values for some variables of a problem because some constraints cannot be satisfied otherwise [48]

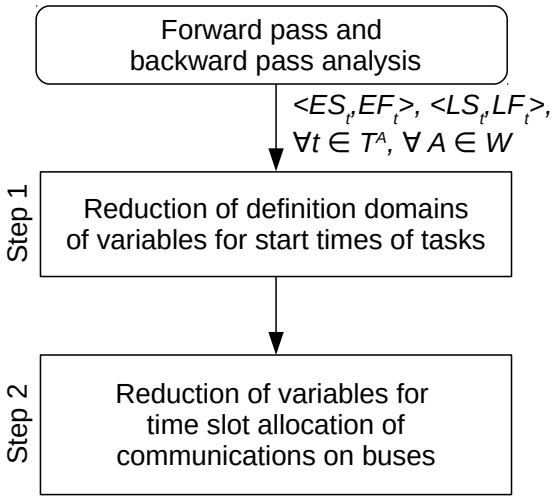


Figure 4.2: An overview of the reduction method.

(see Example 4.3.1). These values are said *inconsistent* with the given constraints. The second step completely removes a subset of time slot variables created for the scheduling of communications.

Example 4.3.1. Let $P = \langle X, D, C \rangle$ be a CSP involving a set of three variables $X = \{x_1, x_2, x_3\}$, their domains $D = \{D_1, D_2, D_3\}$ such that $D_1 = D_2 = D_3 = \{1, 2, 3\}$, and constraints $C = \{c_1, c_2\}$, such that $c_1 \equiv x_1 = x_2$ and $c_2 \equiv x_2 < x_3$. By analyzing constraint c_2, we notice that value $\{1\}$ for variable x_3 can be removed because there is no value smaller than $\{1\}$ in D_2: value $\{1\}$ for variable x_3 is inconsistent with constraint c_2. Similarly, we can remove value $\{3\}$ from D_2. Removing $\{3\}$ from D_2 in turn leads to inconsistency of value $\{3\}$ in D_1 with respect to constraint c_1. Hence, it can also be removed.

Using the same logic presented in Example 4.3.1, we detect and *remove values from the definition domains of temporal variables created for the scheduling of tasks* (variable $start_t$ in Equation (4.1)) that are inconsistent with precedence constraints (constraint (4.3)). The second step *narrows the scope of time slots on which variables are created for the scheduling of communications* (variable $e_{b,s}^{t_m,t_n}$ in Equation (4.2)). It removes the variables that correspond to time slots where the data transfer cannot occur because of data dependency constraints (constraint (4.4)).

4.3.2 The pre-analysis

The two steps rely on a *pre-analysis* to determine *temporal boundaries* of tasks. These boundaries will allow to eliminate the time intervals during which: 1) a task cannot execute, and 2) a communication cannot take place. The analysis is based on the dependencies between tasks, the characteristics of processing resource where a task can be potentially mapped and the application latest finish (deadline) requirement.

We implemented this pre-analysis based on the Critical-Path Method (CPM) [33]. The latter is an algorithm developed in the 1950s by the US Navy and later widely used for scheduling project activities in many domains (e.g., construction, software development, research projects, engineering). CPM operates on any project with interdependent activities and allows to compute:

- The earliest and latest times for each activity to start and finish without exceeding the project due date;
- The critical path, that is, the longest stretch of dependent activities from start to finish.

In the proposed method, we content ourselves with the first output of the critical path method: the earliest and latest times for each activity to start and finish. An activity corresponds to a task in our case. Thus, these boundaries are, for a task t, the earliest start time (ES_t), the earliest finish time (EF_t), the latest start time (LS_t) and the latest finish time (LF_t) (Figure 4.3).

To determine the earliest and latest start and finish times for each task, we reuse two algorithms from CPM: the *forward pass* and the *backward pass*, that we slightly adapt to our problem as follows.

Forward Pass The forward pass algorithm calculates for each task $t \in T^A$, of a given application A, the tuple $\langle ES_t, EF_t \rangle$. ES_t is computed based on input dependencies: a task t cannot start before *all* its predecessors have finished. Let t be a task and $\{t_1, \ldots t_k\}$ the set of predecessors of task t. ES_t is calculated as follows:

$$ES_t = \max_{i \in \{1,\ldots,k\}} \{EF_{t_i}\} + 1 \tag{4.5}$$

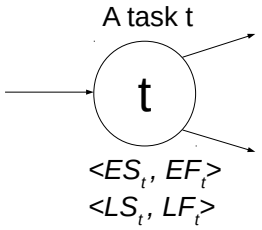


Figure 4.3: A task from the DAG annotated with temporal boundaries.

The forward pass starts exploring an application graph G^A at the source task, so that ES_{source} is assumed to be equal to ES_A. Then, the algorithm browses G^A forward: once ES_t is computed for task t, EF_t is calculated using the following formula where $et(t, c)$ is the execution time of t on a PE c. Note that if a task starts at a given time slot s, this slot is used by the task. Hence, we subtract 1 from the execution time of the task.

$$EF_t = ES_t + \min_{c \in F(t)} \{et(t, c)\} - 1 \tag{4.6}$$

We select the PE c where the execution of t is the fastest. In fact, the forward pass looks for the *earliest* start and finish times and these are obtained when the fastest PEs are used.

Remark. ES_t for a task t corresponds to the case where task t starts executing as early as possible. This case corresponds to a situation where the time to transfer input data to t is equal to zero, i.e., task t and the predecessor(s) having the maximum earliest finish time are mapped to the same PU. Otherwise, feasible solutions could be eliminated. Therefore, Equation (4.6) is simplified by accounting only for execution times.

Backward Pass The backward pass algorithm calculates for each task $t \in T^A$, of a given application A, the tuple $\langle LS_t, LF_t \rangle$. LF_t is first computed based on output dependencies. A task t must complete before the soonest LS of all its successors, so that the application's latest finish time (LF_A) is not missed. Let t be a task and $\{t_1, \ldots t_u\}$ the set of successors of task t. LF_t is calculated as follows:

$$LF_t = \min_{i \in \{1,\ldots,u\}} \{LS_{t_i}\} - 1 \tag{4.7}$$

The backward pass starts exploring an application graph G^A at the sink task, so that LF_{sink} corresponds to LF_A. Then it browses G^A backward until the source. Once LF_t is computed for task t, LS_t is calculated using the following formula:

$$LS_t = LF_t - \min_{c \in F(t)} \{et(t,c)\} + 1 \tag{4.8}$$

Here as well, we select the PE c where the execution of t is the fastest, as it provides the *latest* start and finish times. Similarly to the forward pass presented before, the latest finish time LF_t of a task t corresponds as in the forward pass to the case where communication time between task t and its successors is equal to zero. Thus, Equation (4.8) is also simplified by only capturing execution times.

4.3.3 The reductions

First Step Reduction The scheduling solution selects a start time $start_t$ for each task $t \in T^A$, for each application A in workload $\mathcal{W}$ being studied. This start time is selected from the respective definition domain, initially $\{ES_A, \ldots, LF_A\}$ in Equation (4.1). The first step reduces the number of values that variables $start_t$ can take using the temporal boundaries calculated for tasks to:

$$\forall A \in \mathcal{W}, \forall t \in T^A, start_t \in \{ES_t, \ldots, LS_t\} \tag{4.9}$$

Second Step Reduction Each data transfer is scheduled on communication resources using variables $e_{b,s}^{t_m,t_n}$, Equation (4.2). The variables allocate communication resources to data transfers during time slots. This reduction narrows the scope of time slots interval on which variables $e_{b,s}^{t_m,t_n}$ are defined to the interval on which the data transfer can actually take place: namely, on time slots which follow the earliest finish of the producer and precede the latest start of the consumer. Outside of this interval, time slot allocation variables are not created.

$$\begin{aligned} &\forall A \in \mathcal{W}, \forall c_{t_m,t_n} \in C^A,\ \forall b \in B,\ \forall s \in \{EF_{t_m} + 1, \ldots, LS_{t_n} - 1\}, \\ &e_{b,s}^{t_m,t_n} \in \{0, \ldots, bw_b\} \end{aligned} \tag{4.10}$$

This optimization reduces as well the number of all constraints which apply on variables $e_{b,s}^{t_m,t_n}$ (e.g., Constraint (4.4)).

4.3.4 Example

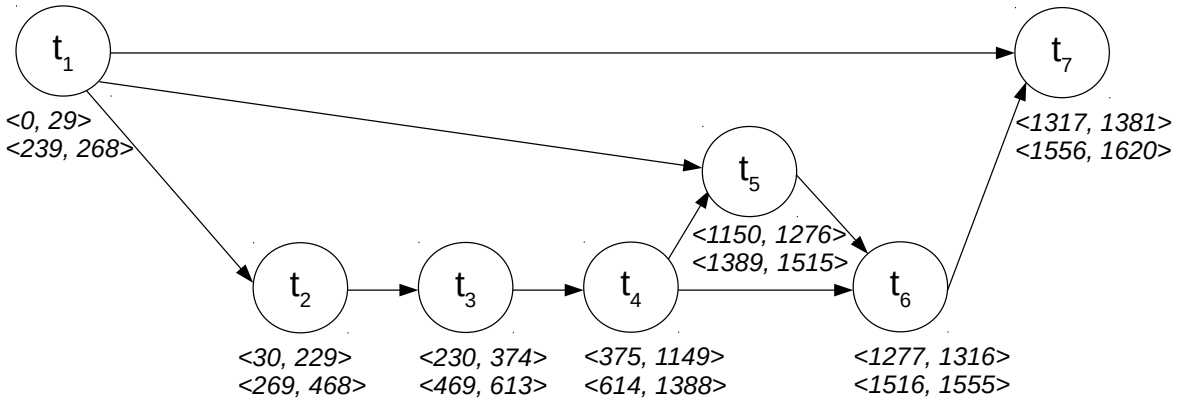


Figure 4.4: An example of an application A where each task $t \in T^A$ is annotated with the tuples $\langle ES_t, EF_t \rangle$ (top annotation) and $\langle LS_t, LF_t \rangle$ (bottom annotation).

Figure 4.4 shows an example of an application that we call A, where tasks are annotated with the temporal boundaries produced by the pre-analysis. Here, we assume that the deadline of the application is equal to 1620 slots and that the application can start running at time slot 0 (i.e., $LF_A = LF_{t_7} = 1620$ and $ES_A = ES_{t_1} = 0$). We assume that a set of three PEs P_X, P_Y and P_Z are available candidates to run the application tasks. Table 4.1 summarizes execution times of tasks on PEs where they can be mapped.

As explained above, the execution time on the fastest unit is selected when the task can be mapped on more than one PE. For example, for tasks t_1, t_4 and t_7 we use respectively the values of 30, 775 and 65 to calculate EF_{t_1}, LS_{t_1}, EF_{t_4}, LS_{t_4}, EF_{t_7} and LS_{t_7}.

From the temporal boundaries calculated and displayed on Figure 4.4, we give few examples of the reductions in Table 4.2.

Table 4.1: Execution times of application A tasks on 3 PEs.

Task	Execution times (slots)		
	P_X	P_Y	P_Z
t_1	150	30	—
t_2	200	—	—
t_3	—	145	—
t_4	2040	—	775
t_5	127	—	—
t_6	—	40	—
t_7	213	—	65

Table 4.2: Examples of the reductions applied on application A from Figure 4.4.

Variable	Before Reduction		After Reduction	
	Scope	Domain	Scope	Domain
$start_{t_5}$	t_5	$\{0, \ldots, 1620\}$	t_5	$\{1150, \ldots, 1389\}$
$e_{b,s}^{t_1,t_2}$	$c_{t_1,t_2},\ \forall s \in \{0, \ldots, 1620\}$ $\forall b \in B$	$\{0, \ldots, bw_b\}$	$c_{t_1,t_2},\ \forall s \in \{30, \ldots, 268\}$ $\forall b \in B$	$\{0, \ldots, bw_b\}$

4.4 Evaluation

In this Section, we evaluate the efficiency of the reduction method applied to the DSE formulation of Chapter 3. We use the same set of four real dataflow application used for evaluations in Chapter 3 from [50] (Figure 3.11). Similarly to Chapter 3 and to [50], in all experiments of this Section, we assume a data packet has a size of 8 data units. We also set a timeout of 1800 seconds after which we stop the exploration if no solution was found. We also assume that execution times given in Figure 3.11 are applicable only to generic processing resources. As in Chapter 3, when tasks are allocated to DSPs or hardware accelerators, their execution times correspond respectively to 1/5 and 1/50 of the times reported.

We conduct four sets of experiments. Experiment 1 and Experiment 2 consider the real-time design objective (deadline-aware solution) on latency while Experiment

3 and Experiment 4 consider the lowest-latency design objective (optimal solution). When multiple applications are considered in parallel in a workload, the optimal solution in this Section refers to a solution for which the sum of latencies is minimal, as expressed in Objective function (3.22), such that the weights for each application are equal to 1. Note that this is an example of how latencies can be minimized using our formulation and that it is of course possible to minimize the latency of the overall workload, or of a subset of applications as explained in subsection 3.6.3.

The four experiments are summarized as follows.

- Experiment 1 first evaluates the solver run-time before and after applying the reduction method. It considers a set of workloads for all combinations of the applications as in Chapter 3. The experiment also evaluates the run-time needed to perform the pre-analysis. We also evaluate the influence of the reduction method on the problem size. The aim is to evaluate the extent of the reductions on the size of the SMT problem model.

- Experiment 2 studies the impact of the reduction method on the finest achievable granularity of the exploration. It considers the heaviest workload of our testbench (the four applications running in parallel).

- Experiment 3 evaluates the run-time to find the lowest-latency (optimal latency) solutions relying on a complementary Binary Search (BS) algorithm for the same set of workloads considered in Experiment 1.

- Experiment 4 evaluates the run-time on different architectures at two levels of slot granularity and shows how granularity can be adjusted to allow solving larger problems sizes. It also provides the optimality gap, in terms of latency, between solutions found at a coarser grain slot and solutions found at the finest granularity.

The target architecture in Experiment 1-Experiment 3 is Architecture A, used in Chapter 3, Figure 3.16.

4.4.1 Experiment 1: Influence on exploration run-time

In this experiment, we consider a set of workloads for all combinations of the applications from Figure 3.11. We perform DSE to find a solution that respects the deadline design objective on latency of each application from the workload. First, we evaluate the speed-up, brought by the reduction technique, by comparing solver run-time when reductions are applied, to previous results found in Chapter 3.

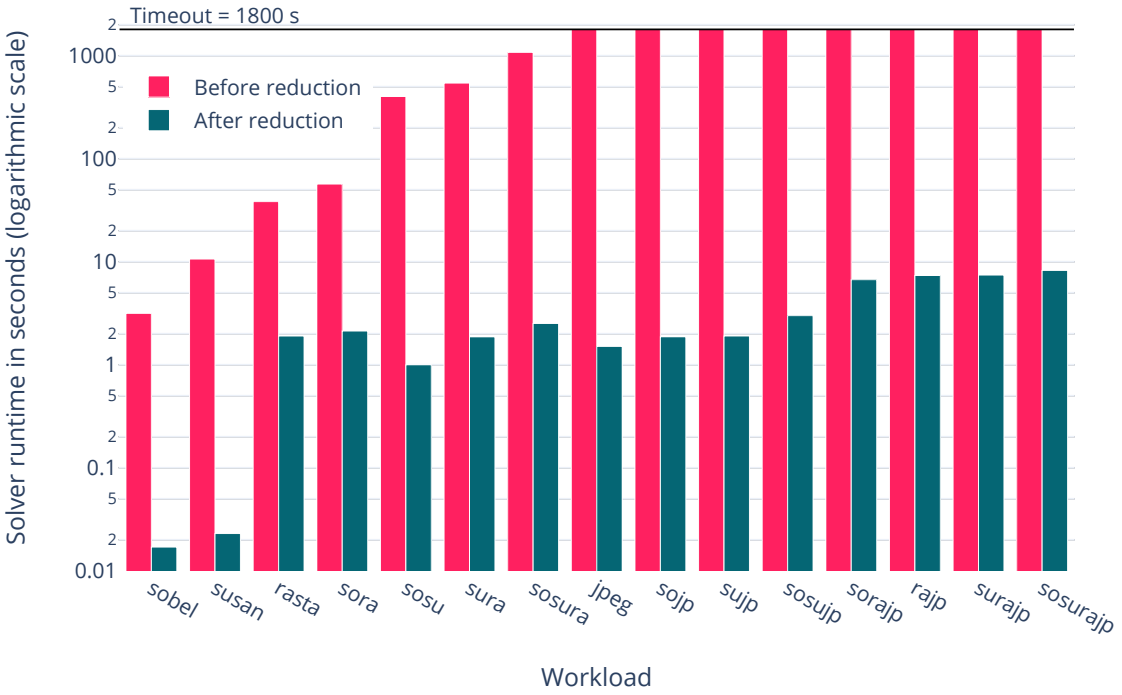


Figure 4.5: Solver run-time to produce a deadline-aware solution, as a function of different workloads using initial formulation and optimized formulation, 1 slot = 1 cycle.

Figure 4.5 shows run-time evolution with a *logarithmic scale*. It can be seen that the reduction decreases the run-time by orders of magnitude. The pink bars indicating run-time before applying reductions show that the solver starts to violate the timeout with workload *JPEG*. This violation results in the impossibility to study 8 out of 15 workloads within the fixed timeout. In contrast, the green bars show the

benefits of our reduction method. The solver run-time tends to increase as well with the increase of workloads complexity but it remains far below the timeout. The run-time values vary in the range $[10^{-2}, 10]$ seconds. This demonstrates the effectiveness of our reduction techniques to reduce the solver run-time and improve the scalability of the exploration process.

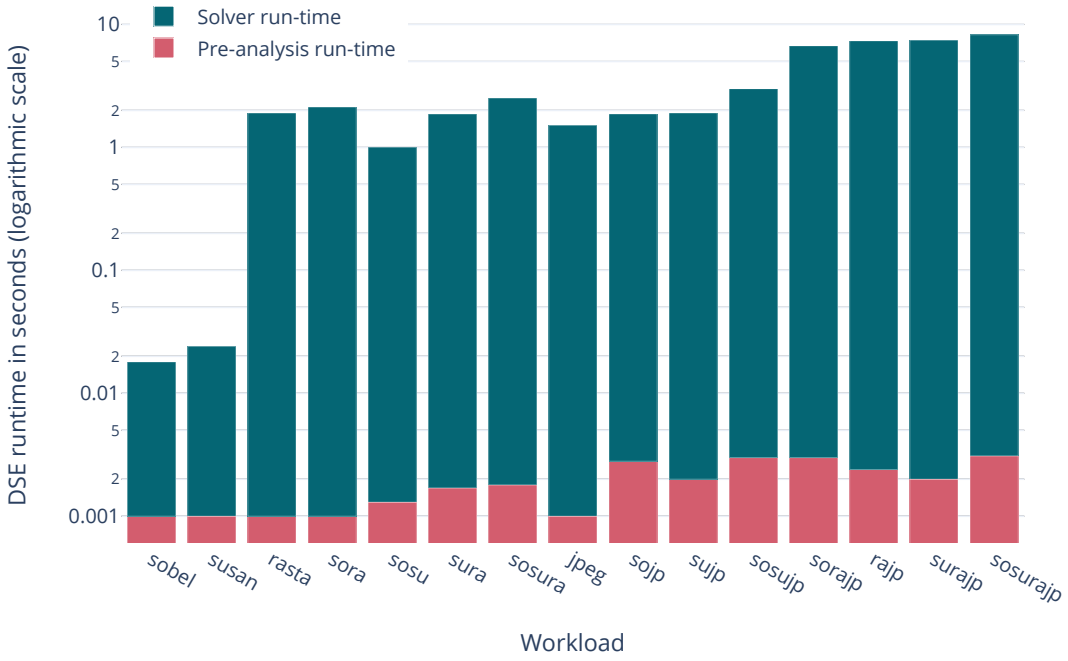


Figure 4.6: Pre-analysis run-time against the solver run-time to produce a deadline-aware solution, as a function of different workloads using optimized formulation.

In this experiment, we also evaluate the practical complexity of performing the pre-analysis (forward pass and backward pass in Section 4.3) that allowed to calculate the temporal boundaries for the reductions. Figure 4.6 shows, for all workloads, the time needed to perform the pre-analysis and generate the tuples $\langle ES_t, EF_t \rangle$, $\langle LS_t, LF_t \rangle$ for each task t, and the solver run-time. Here, a logarithmic scale is used too because the values for the pre-analysis cannot be visualized together with the

Figure 4.7: Comparison of the size of the SMT model, as the number of created variables and constraints, before and after applying the reduction method, as a function of different workloads.

solver values otherwise. The results show that the pre-analysis run-time is relatively negligible compared to the solver run-time.

In this experiment, we also evaluate the influence of the reduction method on the size of the SMT model (number of variables and constraints). In fact, most optimization software algorithms have a practical upper limit on the size of models they can handle, due to either memory requirements or numerical stability [65]. The aim of this evaluation is first to measure the extent of the reduction on the size of the problem and second, to explain the improvement on run-time values given in Figure 4.5. The size of the combinatorial problem has in fact a direct impact on the effort needed

for its resolution.

Figure 4.7 shows the variation of the number of SMT elements for all 15 workloads before and after applying the reduction method. We can see from the figure that the reduction method significantly decreases the size of the SMT model. Focusing only on workload *sosurajp*, we noticed that the size of the problem was reduced drastically: the number of variables decreased from about 130K to 3K and the number of constraints decreased from about 230K to 17K.

4.4.2 Experiment 2: Influence on granularity

As we explained in Section 3.6 of Chapter 3, the length of a slot, l_{slot}, determines the granularity of the analysis, thereby, impacts the precision and the computational effort of the exploration. Reducing the length of the slot leads to a more accurate analysis of the mapping and scheduling problem. However, it involves creating more slot-related variables, which translates to a bigger size of the problem and a longer time needed to solve it.

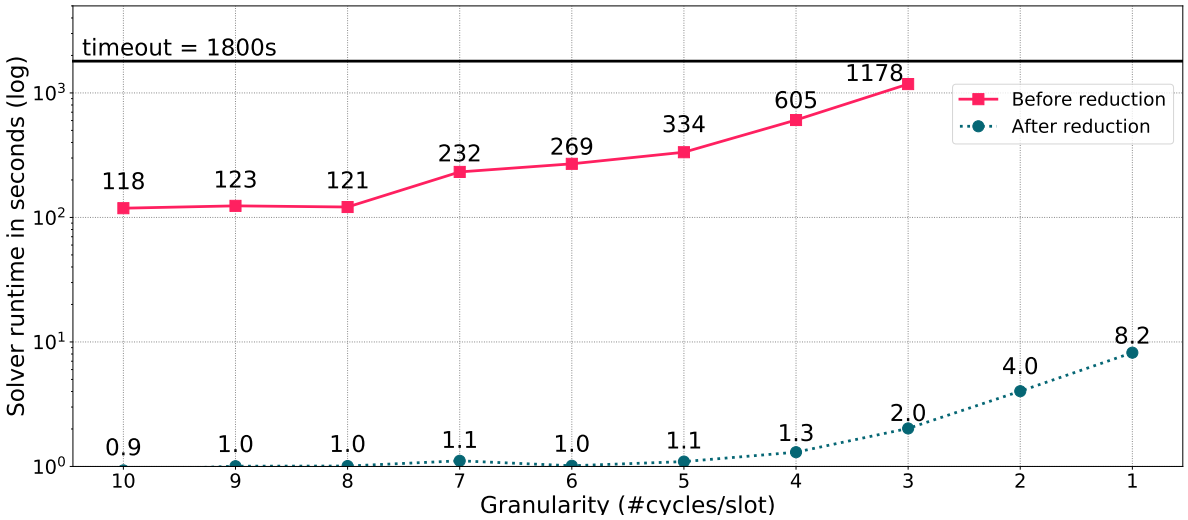


Figure 4.8: Solver run-time to produce a deadline-aware solution, as a function of exploration granularity using initial formulation and optimized formulation for workload *sosurajp*.

Figure 4.8 shows the solver run-time for a fixed workload, *sosurajp*, as a function of l_{slot} that varies from 10 cycles down to 1 cycle. This curve represents the impact

of the DSE granularity on the solver run-time and allows to appreciate the influence that our reductions have on that.

By considering the same timeout of 1800 seconds, the curve plotting run-time evolution without our reduction method shows that it is not possible to explore the design space with a granularity smaller than 3 cycles/slot. On the contrary, the curve plotting results of the optimized formulation shows that the latter allows to study the workload up to the maximum precision (l_{slot} = 1 cycle) within only 8.2 seconds.

This experiment demonstrates that the reduction method, by improving the scalability of the DSE approach, also allows a more accurate analysis of the system under study. This helps reducing the gap between the estimations of the SMT model and the actual behavior of the system leading to more realistic exploration results.

4.4.3 Experiment 3: Optimal solution search

In this experiment, we investigate if the reductions will enable to find the lowest-latency solution. As the solver is mostly efficient in finding satisfiable solutions, we rely on a complementary Binary Search (BS) algorithm to find optimal solutions.

The BS is an algorithm commonly used to find the position of a given element in a sorted range of elements (list). It looks at the middle element of the list, compares it with the sought element, and continues the search in the relevant half of the list until the sought element is found. Assuming a list of size n, in the worst case, the BS algorithm visits at most $\log(n)$ elements of the list before finding the sought element: it has a worst-case running time of $O(\log\ n)$.

We apply the BS on the range of time slots that are comprised between the application earliest finish time and its latest finish time, which are determined by the pre-analysis of Section 4.3.2. The minimum latency $minlat$ is the sought element and it is identified by these two conditions:

1. The problem is satisfied for: $latency \leq minlat$

2. The problem is unsatisfied for: $latency \leq minlat - 1$

Here, when multiple applications are considered simultaneously in a workload, latency refers to the sum of latencies of all applications with all weights equal to 1.

Figure 4.9 shows run-time values to find the optimal solution using a BS and the optimized formulation, against run-time values to find *a first deadline-aware solution using the initial formulation*. Note that we do not compare against run-time values to find optimal solutions using the initial formulation since the considered timeout is exceeded even to find a first feasible solution for the initial formulation.

From Figure 4.9, we can see that while the initial formulation fails to find a first deadline-aware solution for 8 workloads, the reductions allowed to find optimal solutions for all workloads within the time limit. From the figure, it can be seen that the time to find the optimal solution using a BS and the optimized formulation varies between 1.08 seconds (*susan*) and 89.98 seconds (*sosurajp*).

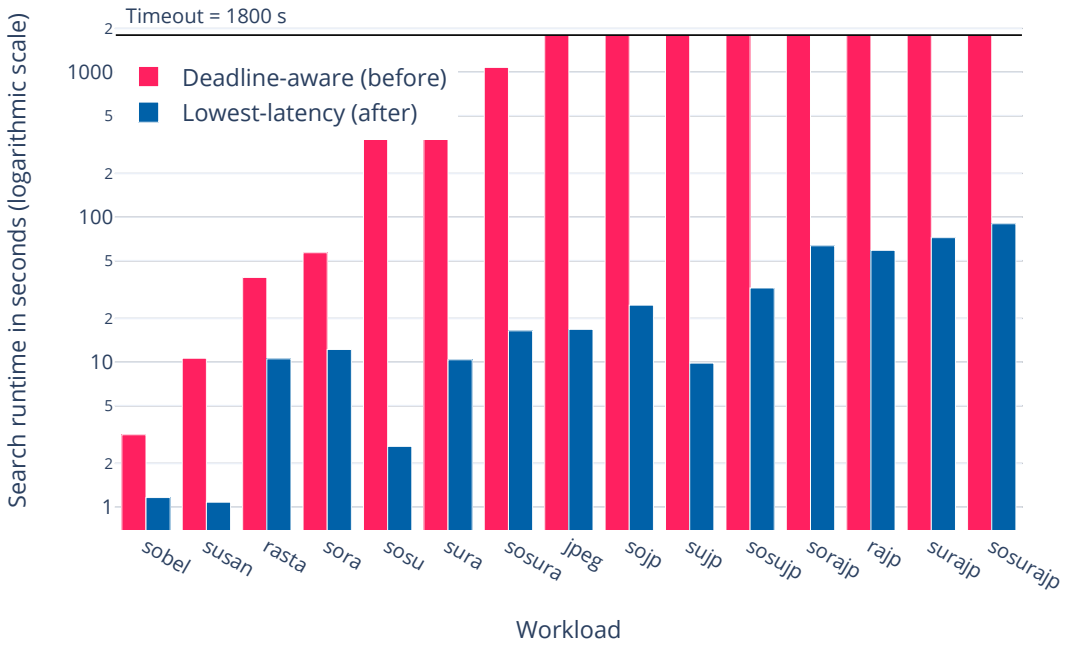


Figure 4.9: Search run-time to produce an optimal solution using the optimized formulation and a complementary BS, against run-time values to find a first deadline-aware solution using the initial formulation, as a function of different workloads.

Thus, in this experiment, we have showed that the reductions, beyond reducing significantly the run-time values to find deadline-aware solutions, have enabled to

reach optimal solutions within the time limit for all workloads, whereas the initial formulation fails to find a first deadline-aware solution for 8 workloads.

4.4.4 Experiment 4: Adjusting granularity to solve larger scale problems

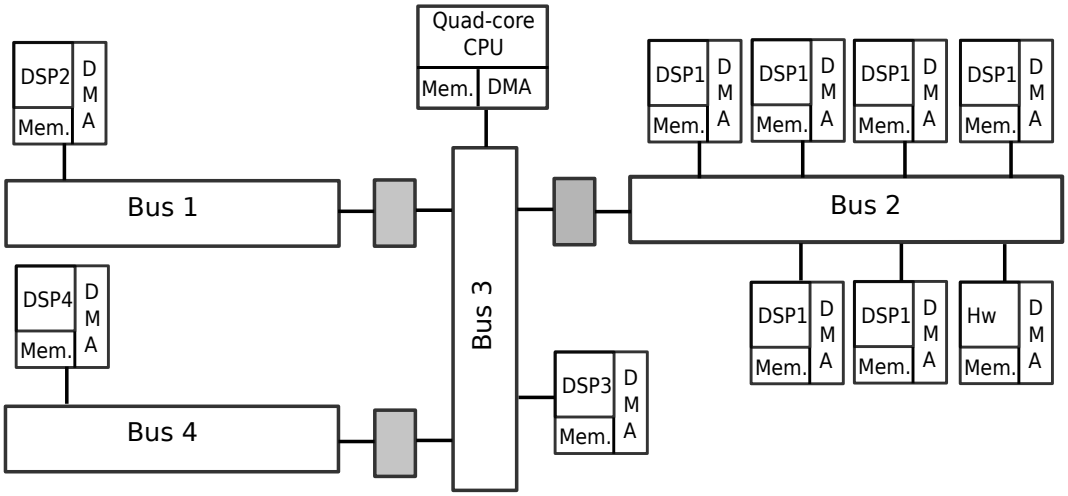


Figure 4.10: Block diagram of Architecture B used in Experiment 4.

As shown in Experiment 2, adjusting the slot granularity impacts the search run-time: For the initial formulation, a coarser grain slot allowed to solve a problem that was not solvable, within the timeout, at a finer grain slot. For the optimized formulation, all workloads are solvable even at the finest granularity (1 slot = 1 cycle), but a coarser grain slot reduces the run-time.

In this experiment, we leverage this property of our modeling to investigate, for the optimized formulation: 1) How granularity can be adjusted to allow solving larger problems sizes, and 2) The optimality gap, in terms of latency, with respect to the solution found at the finest granularity.

We consider 4 different architectures illustrated in Figures 4.10, 4.11, 4.12 and 4.13. Architecture C differs from Architecture B by the deployment of an additional quad-core CPU (4 additional PEs). Architecture B, Architecture C and Architecture E have the same set of PUs, but with a different interconnect.

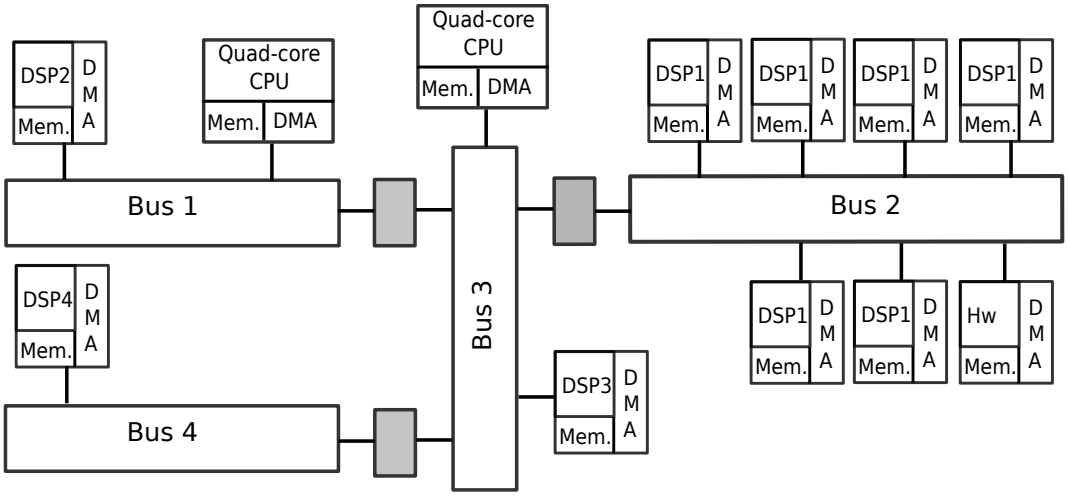


Figure 4.11: Block diagram of Architecture C used in Experiment 4.

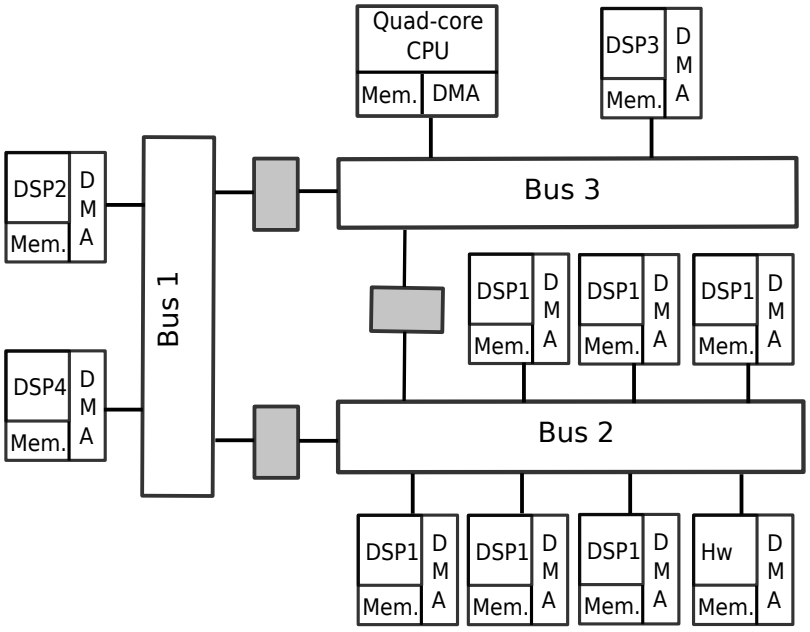


Figure 4.12: Block diagram of Architecture D used in Experiment 4.

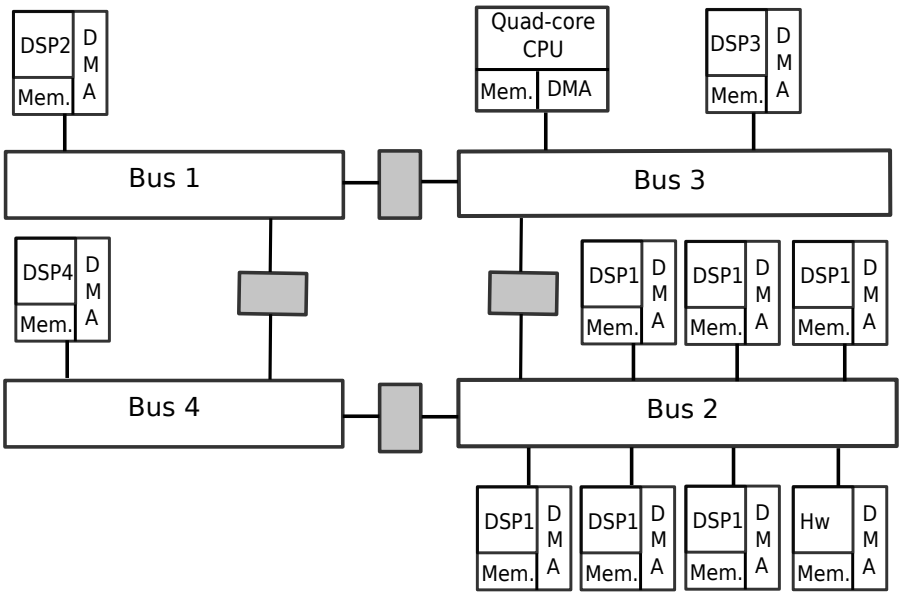


Figure 4.13: Block diagram of Architecture E used in Experiment 4.

Figures 4.14, 4.15, 4.16 and 4.17 show, for the 4 architectures, the evolution of run-time to find lowest-latency solutions, for the set of all workloads from previous experiments, when the granularity is set to: 1) $granularity = 1$, i.e., $1\ slot = 1\ cycle$ and 2) $granularity = 3$, i.e., $1\ slot = 3\ cycles$.

From the figures, it can be seen that the coarser granularity (i.e., $1\ slot = 3\ cycles$) allows to reduce run-time for all workloads, on all architectures. In Figure 4.15, the run-time reduction due to a coarser granularity is particularly appreciated as it allowed to solve 7 workloads that were not solvable within the timeout, at a finer granularity.

Table 4.3 shows the optimality gap and the speed-up factor of solutions found at $granularity = 3$, compared to solutions found at $granularity = 1$, for the set of all workloads, on the 4 different architectures. Note that for solutions that we were not able to solve within the timeout, at $granularity = 1$, we exceptionally exceed the timeout to be able to evaluate the optimality gap and the speed-up factor. The aim here is to evaluate the solutions that we could find within the timeout, at a coarser grain slot.

It shows that, on average, (for all workloads, on all architectures) the 3 times coarser granularity brings a 4.83 times speed-up. From the Table, we can also cal-

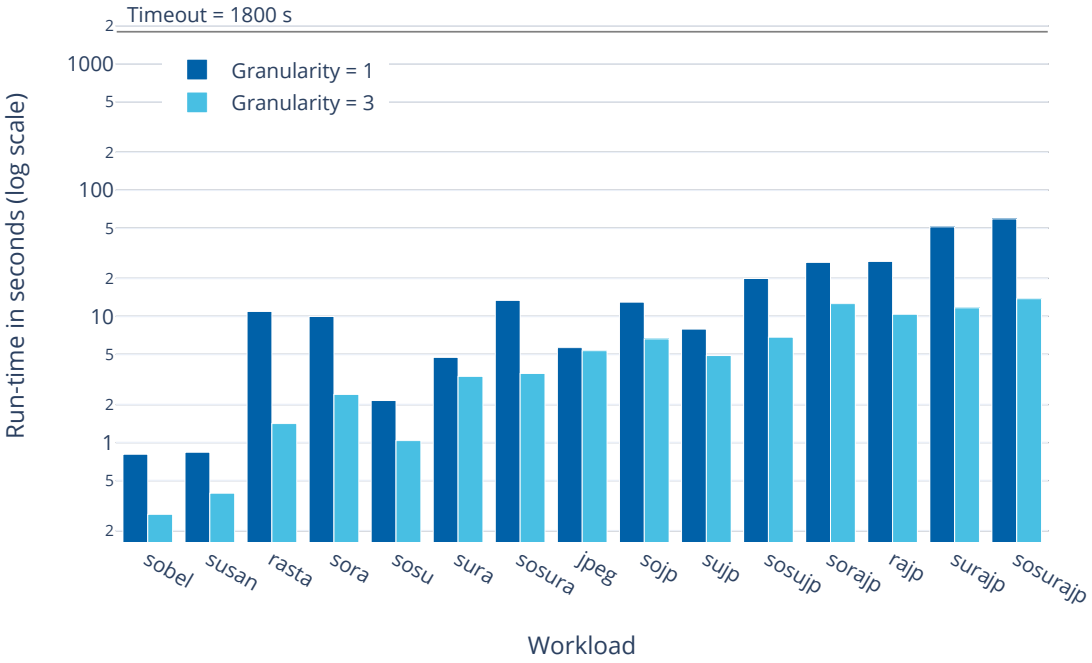


Figure 4.14: Run-time to search the minimal latency solution on Architecture B using two levels of granularity (1 and 3), as a function of different workloads.

culate an average optimality gap equal to only 0.25%. The speed-up is calculated using the following formula: $\frac{runtime(granularity=1)}{runtime(granularity=3}$).

This experiment shows that, on our benchmark, reducing the precision by varying the granularity from 1 to 3 allows to: 1) find near-optimal solutions faster for all studied problems, and 2) find near-optimal solutions for larger problems that we were not able to solve within the timeout, at $granularity = 1$. Varying the granularity can mitigate the practical limitation of the proposed approach. However, the precision of estimations, and thereby, the optimality gap depends on the quantity and significance of information being missed within the chosen smallest observable time slot. In fact, when the mapping and scheduling of tasks and transfers is more likely to span over long durations, more time slots are necessary, so the solver takes longer to find a suitable or an optimal solution. Obviously, bigger time slots help handling this issue. However, larger time slots can prevent the analysis of delays below the

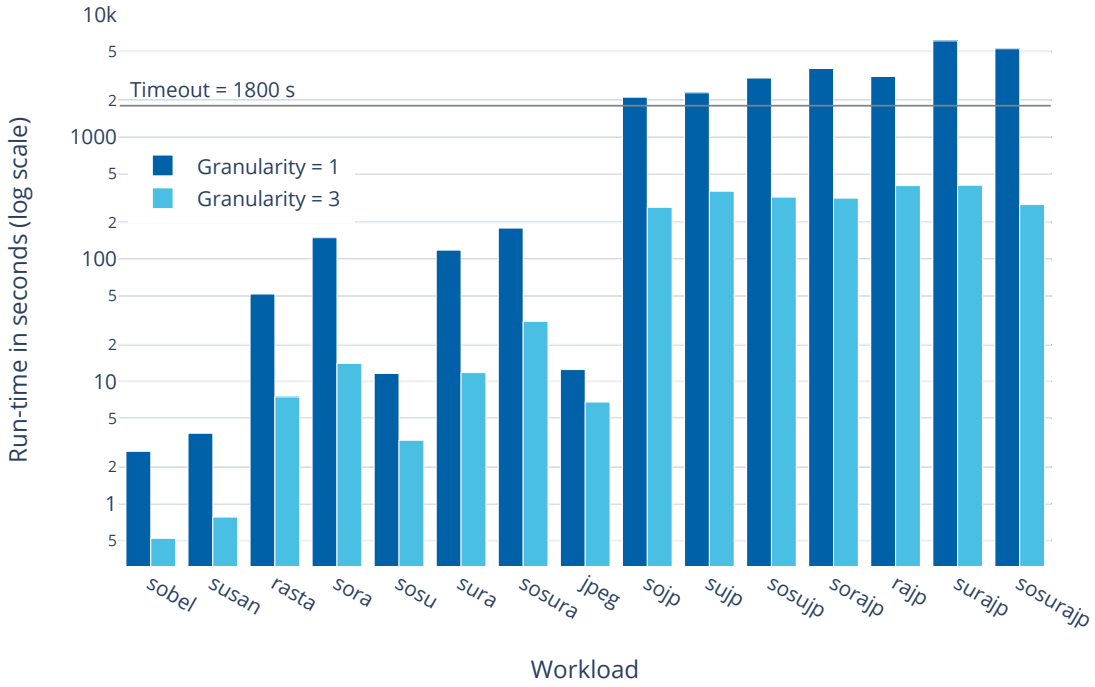


Figure 4.15: Run-time to search the minimal latency solution on Architecture C using two levels of granularity (1 and 3), as a function of different workloads.

length of the time slot. This loss of precision can be particularly significant in problems where timing information (e.g., duration of tasks) is disparate (e.g., different orders of magnitude)

4.5 Conclusion

In this Chapter we presented a technique to prune the design space of tasks and communications scheduling problem and reduce considerably the overall DSE run-time. A set of experiments showcased the benefits of our reduction method: First, we showed the important speed-up of the SMT solver run-time to solve the optimized formulation. For our testbench, the speed-up varies from 20 up to 589. Then, we showed that the reductions enabled to analyze larger problems more accurately by

Table 4.3: Optimality gap (%) and speed-up factor of solutions found at granularity = 3, compared to solutions found at granularity = 1, for a set of workloads.

workload	gap (%)	speed-up
sobel	0.21	2.99
susan	0.09	2.11
rasta	1.30	7.73
sora	0.59	4.15
sosu	0.12	2.08
sura	0.36	1.41
sosura	0.23	3.79
jpeg	0.05	1.06
sojp	0.09	1.95
sujp	0.07	1.62
sosujp	0.09	2.92
sorajp	0.18	2.12
rajp	0.25	2.62
surajp	0.14	4.38
sosurajp	0.10	4.29

(a) Architecture B.

workload	gap (%)	speed-up
sobel	0.63	5.15
susan	0.88	4.84
rasta	0.19	6.94
sora	0.20	10.67
sosu	0.12	3.52
sura	0.12	10.04
sosura	0.05	5.80
jpeg	0.05	1.84
sojp	0.09	7.95
sujp	0.07	6.39
sosujp	0.03	9.42
sorajp	0.04	11.49
rajp	0.08	7.81
surajp	0.09	15.16
sosurajp	0.00	18.82

(b) Architecture C.

workload	gap (%)	speed-up
sobel	0.21	2.50
susan	0.09	2.94
rasta	1.30	8.05
sora	0.59	6.35
sosu	0.12	3.16
sura	0.36	2.10
sosura	0.23	4.61
jpeg	0.05	1.59
sojp	0.09	1.80
sujp	0.07	1.45
sosujp	0.09	2.64
sorajp	0.18	2.40
rajp	0.25	4.74
surajp	0.14	9.72
sosurajp	0.10	7.18

(c) Architecture D.

workload	gap (%)	speed-up
sobel	0.21	2.69
susan	0.26	1.68
rasta	1.30	11.14
sora	0.59	4.31
sosu	0.12	3.55
sura	0.60	6.83
sosura	0.42	3.81
jpeg	0.05	1.36
sojp	0.09	2.60
sujp	0.07	2.71
sosujp	0.03	1.11
sorajp	0.18	4.09
rajp	0.25	3.81
surajp	0.26	4.14
sosurajp	0.20	3.42

(d) Architecture E.

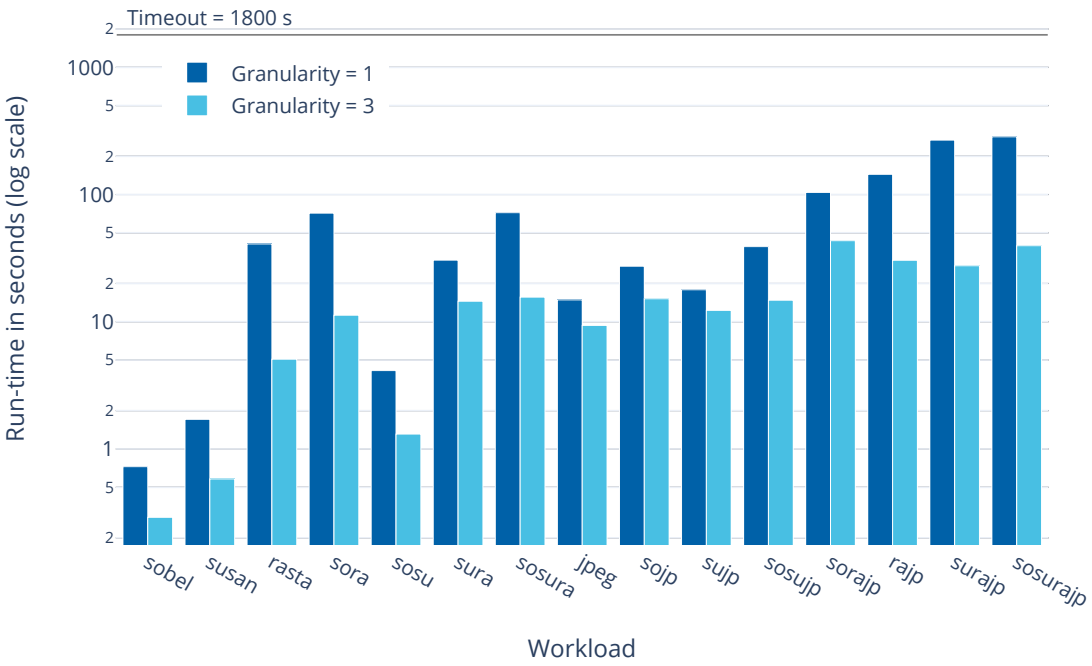


Figure 4.16: Run-time to search the minimal latency solution on Architecture D using two levels of granularity (1 and 3), as a function of different workloads.

allowing their exploration at finer granularity levels. Moreover, we showed that the reductions allowed to reach optimal solutions for *all* workloads, while, before the reductions, the solver fails even to find deadline-aware solutions for more than half the workloads. Finally, we showed that the practical limitation of our approach can be mitigated thanks to the adjustable granularity of time slots with an average optimality gap as small as 0.25% for granularity equal to 3.

The method proposed in this Chapter broadens the applicability of SMT-like approaches (e.g., CP, ILP), to solve the scheduling of tasks and communications in a holistic manner, for larger problem sizes.

In the next Chapter, we study the extension of the optimized SMT formulation, with a new module describing power consumption of the interconnect. This is facilitated by the adoption of a CP-based approach, characterized with modularity and extensibility. Moreover, the analysis of power consumed by the interconnect could

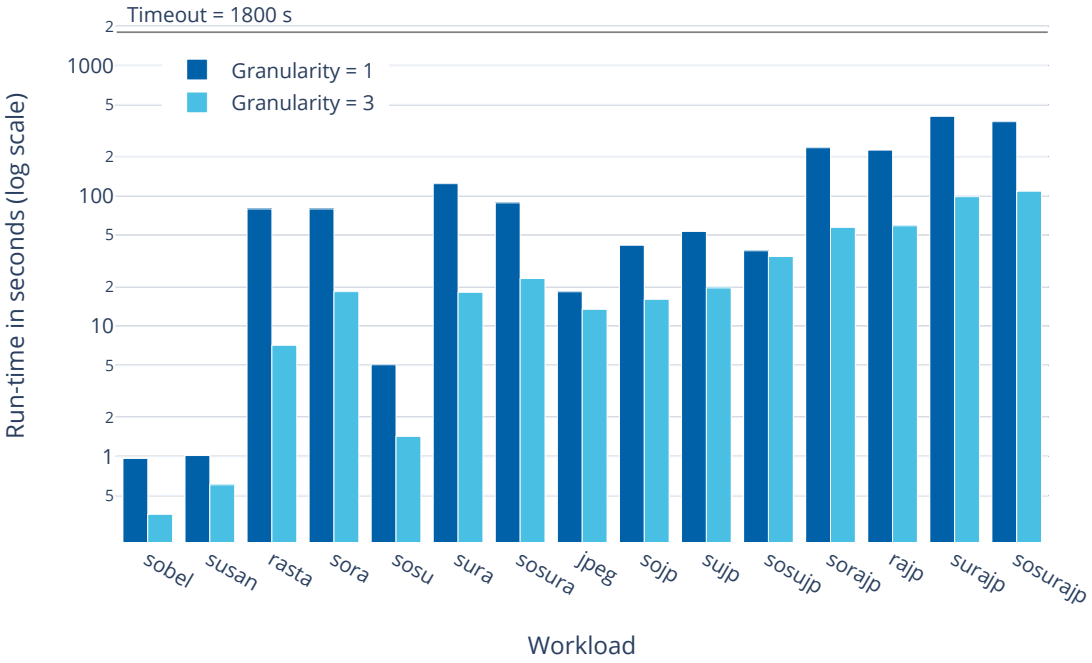


Figure 4.17: Run-time to search the minimal latency solution on Architecture E using two levels of granularity (1 and 3), as a function of different workloads.

benefit from the reductions presented in this Chapter, as these allow to identify the time domains on which the interconnect is actually used to perform communications, and ignore time domains where communications cannot be performed. As such, the power consumption analysis can be focused only on time domains eligible to communications.

Chapter 5

Extending the Latency SMT Model with a Power Consumption Model for Multi-Bus Interconnects

5.1 Introduction

In previous Chapters, we provided a Design Space Exploration (DSE) approach for multi-bus architectures with a latency design objective. The latter consists either in minimizing latency or satisfying a given threshold. However, in real systems, it is often required to consider multiple, and often conflicting design objectives which cannot be completely and/or simultaneously achieved: Optimizing one criterion degrades the other criterion and vice versa. In this Chapter, we address this limitation by studying the extension of our Constraint Programming (CP)-based DSE approach, with a second design objective. The purpose of this Chapter is to study the capabilities and limitations of the proposed DSE to address a problem with more than one conflicting design objective.

As discussed in Chapter 2, one of the main advantages of the CP approach is

the ability to extend it easily. This is enabled by two properties of the CP approach: separation of concerns and modularity. By separating the problem definition part from the resolution engine, different problem models can be solved without changing the resolution engine. Moreover, to extend an existing CP model, new modules (i.e., decision variable, constraints, design objectives) defining additional properties of the solution to be found (e.g., the proposed solution should be power-efficient) can be plugged into the initial model without significant modification of existing modules, nor the resolution engine.

Since in this thesis we have focused on multi-bus architectures, we will consider the power consumption of multi-bus interconnects as a second design objective in this study. The interconnect power module proposed in this chapter can be extended with other modules to account for the power consumption of processing components. Several models from the literature allow to analyze power consumption of processing resources, see for example [51].

In this chapter, we study the bi-criteria problem for the optimization of latency and power consumption[1]. In fact, latency and power consumption are two conflicting design criteria: We cannot obtain a single solution that optimizes both of them simultaneously and completely. For instance, in order to reduce power consumption, we need to reduce the supply voltage and the operating frequency, which increases the latency. Thus, there might be multiple solutions that satisfy different levels of satisfaction for each criterion, and these solutions can be produced using different methods. A detailed survey on methods for multi-criteria (objective) optimization problems can be found for example in [40]. These methods are often classified according to the implication of the *decision maker* in the solution process [42]. The decision maker is a person who is able to express preference information about conflicting criteria. In our case, it refers to a designer. In *a priori methods*, the designer first manifests his/her preference, subsequently, a solution which satisfies his/her requirement is searched. In *a posteriori methods*, a set of Pareto optimal solutions are generated first, then the designer selects the most preferred design. Here, the designer does not have to express his/her preference before the solution process. While the second class may involve a more computationally expensive search process, the first class

[1] For the sake of simplicity, in the rest of this manuscript, we will sometimes refer to the power consumption of the interconnect simply by power consumption.

requires the designer to have a fair understanding of the problem possibilities and limitations in advance, which is not always true.

Examples of *a priori methods* are *lexicographic ordering* [24] where defining a priority order between the two criteria allows to address the problem in a hierarchical manner: higher priority criterion is optimized first, at the expense of the other criterion. Thus the problem is transformed into two single-objective problems which are solved sequentially in accordance with their importance. This priority structure requires that the designer has a priori specified a preference order between the studied criteria. The solution of *lexicographic ordering* can be proven to be Pareto optimal [42]. Another example is the *bounded objective function method* which consists in transforming one criterion into a constraint defining a satisfactory threshold (or a limiting budget), and to minimize the second one under this constraint. For example, we can minimize latency under a certain available power budget, or minimize power consumption under a given satisfactory latency threshold. This method also assumes that the designer has a priori defined an acceptable threshold for one of its criteria.

For *a posteriori methods*, different methods can be used to generate the set of Pareto optimal solutions like the weighted sum method, the Normal-Boundary Intersection (NBI) method [18], the ε-constraint method [80], or Evolutionary Algorithms (EAs).

In this Chapter, we will provide two *a priori* approaches and one *a posteriori* approach to address the latency-power consumption bi-criteria problem: In the first approach, it is assumed that a priority order has been established and that latency is the most important criteria. A second approach minimizes power consumption under a given threshold on latency. A third approach aims at identifying the set of Pareto optimal solutions.

The rest of this Chapter is organized as follows: Section 5.2 introduces the SMT model proposed to estimate power consumption and bandwidth requirements of bus segments. Section 5.3 presents the three approaches proposed to optimize both latency and power consumption. Section 5.4 proposes a series of evaluations of the proposed approaches. Finally, Section 5.5 concludes this Chapter.

5.2 Power Consumption Modeling for the Multi-Bus Interconnect

5.2.1 Scope and assumptions

In a multi-bus architecture, the bandwidth requiremernts on each bus segment can be significantly different [32, 73, 77]. It is therefore needed to identify at which bandwidth each bus should operate. The goal of the model we propose is twofold: First, it allows to capture and minimize the power consumed by the interconnect. Second, it allows to identify, for each bus segment, the best operating bandwidth to achieve latency and power consumption design objectives.

We assume that each bus segment b can be used in a set of operating modes $\mathcal{M}_b$. A mode $M \in \mathcal{M}_b$ determines the parameters of performance and power consumption of the bus. It is denoted as a tuple $< bw_M, power_M >$ where bw_M is the maximum achievable bandwidth when a bus segment is configured to mode M, and $power_M$ is the power consumed by the segment corresponding to mode M. The values of bw_M and $power_M$ are expressed per time unit that corresponds to a single time slot. The definition of a *time slot* is the same as in Chapters 3 and 4. Different modes can lead to different performance and power consumption figures. The aim of the modeling is twofold: 1) Minimize power consumed by the interconnect, and 2) allocate an operating mode to each bus such that latency and power consumption design objectives are met. The modeling is based on the following assumptions:

- The power consumed by a bus, on a time slot, is proportional to the activity (i.e., number of transferred data units) that takes place within the slot duration. For example, for a slot on which only 30% of bw_M is used, a bus consumes only 30% of $power_M$.

- A single operating mode is determined for each bus, over the full period under analysis—We assume that operating mode changes are costly and that they take place only between the periods on which we conduct our analysis. It is determined by the highest bandwidth reached over the entire scheduling period (i.e., on all time slots). For example, if a bus has two operating modes: $M1 =< bw_{M1}, power_{M1} >$ and $M2 =< bw_{M2}, power_{M2} >$ such that $bw_{M2} >$

bw_{M1}. Considering two time slots s_i and s_j where the bus operates respectively at bw_i and bw_j such that $bw_i \leq bw_{M1}$ and $bw_{M1} < bw_j \leq bw_{M2}$. The operating mode of the bus here is assumed to be $M2$.

- The amount of power $power_M$ consumed by a bus on a single time slot, includes the power consumption of all the interconnect circuit and logic (e.g., wires, arbiter logic).

5.2.2 The SMT model

In this modeling, we reuse the pre-analysis presented in Chapter 4 to reduce the domain of time slots, on which power consumption and bandwidth requirements of buses are analyzed, to the set of time slots where communications can actually occur. Therefore, the same assumptions from Chapter 4 apply. We consider the set of eligible slots $S_{\mathcal{C}}$ defined, in Equation (5.1), as the union of $S_{c_{t_m,t_n}}$, for all communications c_{t_m,t_n} in $\mathcal{C}$. We remind that $\mathcal{C}$ is the union of all workload communications, as explained in Section 3.3. For a given data dependency $c_{t_m,t_n} \in \mathcal{C}$, $S_{c_{t_m,t_n}}$ is the set of time slots that lie between the earliest finish time of producer task t_m and the latest start time of consumer task t_n as defined in Equation (5.2).

$$S_{\mathcal{C}} = \{S_{c_{t_m,t_n}} \mid c_{t_m,t_n} \in \mathcal{C}\} \tag{5.1}$$

$$\forall c_{t_m,t_n} \in \mathcal{C}, S_{c_{t_m,t_n}} = \{s \mid s \in \{EF_{t_m} + 1 \ldots LS_{t_n} - 1\}\} \tag{5.2}$$

The power consumed by a single bus segment, $Power_b$, is calculated as the product of its utilization U_b, and the power consumption $Power_b^{S_{\mathcal{C}}}$ corresponding to its full usage on eligible time slots $S_{\mathcal{C}}$.

$$\forall b \in B, Power_b = U_b \times Power_b^{S_{\mathcal{C}}} \tag{5.3}$$

As the value of power consumption corresponding to a mode M, is given per time slot, $Power_b^{S_{\mathcal{C}}}$ is calculated as the product of the power consumption corresponding to the mode M being analyzed, and the cardinal of $S_{\mathcal{C}}$.

$$\forall b \in B, Power_b^{S_{\mathcal{C}}} = |S_{\mathcal{C}}| \times power_M \tag{5.4}$$

The bandwidth utilization U_b of a bus b is defined as the proportion of the maximum available bandwidth of a given mode M, being actually used to transfer data, on the set of eligible slots S_C. It is captured by the following Equation.

$$\forall b \in B, U_b = \frac{1}{|S_{\mathcal{C}}| \times bw_M} \times \sum_{s \in S_{\mathcal{C}}} \sum_{c_{t_m,t_n} \in \mathcal{C}} e_{b,s}^{t_m,t_n} \tag{5.5}$$

Therefore, the expression of power consumption of a single bus segment can be calculated as follows.

$$\forall b \in B, Power_b = \frac{power_M}{bw_M} \times \sum_{s \in S_{\mathcal{C}}} \sum_{c_{t_m,t_n} \in \mathcal{C}} e_{b,s}^{t_m,t_n} \tag{5.6}$$

The bandwidth utilized on a bus does not exceed the bandwidth of the operating mode being analyzed.

$$\forall b \in B,\ \forall s \in S_C, \sum_{c_{t_m,t_n} \in C} e_{b,s}^{t_m,t_n} \leq bw_M \tag{5.7}$$

The power consumed by the entire interconnect is captured by the sum of power consumed by each single bus, as follows.

$$Power_{interconnect} = \sum_{b \in B} Power_b \tag{5.8}$$

In this model, we aim at identifying for each bus, an operating mode that will be used during the execution of all the studied workload. This mode, for a given bus, should be the mode that is able to grant the minimum power consumption of the overall interconnect while satisfying design objectives on latency. However, we do not encode bus operating modes parameters (i.e. bw_M and $power_M$) as decision variables in the SMT formulation, but instead as input constant parameters. As such, each time the solver is invoked to solve this model, a unique operating mode can be actually analyzed for each bus.

It is therefore suggested, to solve the formulation for *all possible combinations* of the pair (bus, operating mode) that are available: Assuming that each bus b is associated with a set of operating modes $\mathcal{M}_b$, the number of combinations to analyze is given by $n_{combinations}$.

$$n_{combinations} = \prod_{b \in B} |\mathcal{M}_b| \tag{5.9}$$

For example, for an architecture with 2 buses, each having 3 operating modes, 9 combinations need to be solved in total. After that, the results of each combination are compared and the combination that provides the minimum power consumption is retained as the optimal power consumption solution. In fact, decomposing the problem by considering the operating mode of each bus segment as a constant input parameter simplifies greatly the SMT problem for two reasons: First, it reduces the number of decisions to be solved, therefore, the memory allocated in the system to solve the model, and second, it avoids forming a complex, hard-to-solve formulation between operating mode variables, and variables $e_{b,s}^{t_m,t_n}$ (see Equation 5.6 for example). The limitation of this approach will mainly come from the number of combinations to analyze: the more combinations of (bus, operating mode) can be formed, the longer will be the exploration run-time, since the solver will be invoked more times.

5.3 DSE for Latency and Power Optimization

The power consumption modeling proposed in Section 5.2 can be used with the formulation presented in Chapter 3, and optimized in Chapter 4, in different ways. In regards to signal processing applications, in order to work properly, these typically have to fulfill some design objectives on latency, that are, an application must be completed within some fixed time budget, or as fast as possible. In this Section, we first present two approaches to answer this requirement. In both approaches, we suggest to minimize power consumption under a certain design objective on latency. The latter is either to satisfy a deadline (threshold) constraint or to minimize latency. Power consumption optimization is performed under the condition that the latency design objectives are fulfilled. Subsequently, we propose a third approach which aims at producing a set of Pareto optimal solutions that a designer can analyze to choose the most preferred design.

5.3.1 Approach 1: Minimize power under minimal latency

In this approach, we suggest to optimize latency and power consumption such that the uppermost priority of the design is to guarantee the lowest latency. Hence, the approach considers latency and power optimization in a hierarchical manner, and

in two phases: The first phase aims at finding minimal latencies for each application. For this, the same SMT modeling and solving of previous Chapters are used. Once, an optimal latency solution is found, a second phase, where the modeling of the initial phase is complemented with the power model, solves the overall problem such that power consumption is minimized *under the minimal latency constraints*. During this second phase, the only restrictions set are on *the value of latency of each application*. This means that *all different design candidates* (e.g., task mapping, communication mapping, slot allocation) with the *same optimal latency per application* are explored. In case the calculation of minimal latencies is based on the minimization of the global latency (i.e., sum of latencies of different applications), several solutions can be found where the global latency is minimal (thus identical), but latencies per application are different. In such a case, we continue to the second phase of power minimization relying on a single solution of latency per application. Other latency solutions could lead to lower power consumption but we assume that the first phase is in charge of fixing the satisfactory latencies, and that the second phase finds minimum power consumption under these latency constraints.

5.3.2 Approach 2: Minimize power under deadline constraints

For this approach, we propose to minimize power consumption while satisfying deadline constraints for applications. A single phase is performed here where the overall SMT modeling (Chapters 3 and 4 + power model, Section 5.2) is used. A satisfy design objective (constraint) is considered for latency, while a minimize design objective is considered for power consumption. *All different design candidates* (e.g., task mapping, communication mapping, slot allocation) that meet the deadlines are explored.

Remark. Note that for the same system under study, it is also possible to combine the two approaches, i.e., minimize power consumption under: deadline constraints for some applications and minimal latency for other applications. Yet, it is also possible to minimize latency under some power budget allocated to the interconnect. In general, from a modeling point of view, any combination of satisfy and/or minimize design objectives on latency and power consumption can be encoded.

5.3.3 Approach 3: Pareto optimality for latency and power consumption

In this subsection, we will first remind the concept of Pareto optimality: In a multi-criteria optimization problem, there are more than one design objective optimizing competing criteria. Therefore, there does not exist a single solution able to optimize all criteria simultaneously and/or completely. The solution to such problems is thereby a set of solutions that define the best trade-offs between conflicting objectives. Here, the goodness of a solution is evaluated according to the Pareto optimality concept. There are two levels of Pareto optimal solutions [42]: *weak Pareto optimum* and *strong Pareto optimum*[2], defined as follows.

Definition 5.3.1 (Weak Pareto optimum)**.** A solution S is called *weakly Pareto optimal* if there does not exist another solution S' such that: S' is *strictly better* than S in *all* objectives.

Definition 5.3.2 (Strong Pareto optimum)**.** A solution S is called *strongly Pareto optimal* if there does not exist another solution S' such that: S' is *no worse* than S in all objectives *and* S' is *strictly better* than S in *at least one objective*.

The set of strongly Pareto optimal solutions is a subset of the set of weakly Pareto optimal solutions. Several methods can be used to produce the entire set of (weakly and/or strongly) Pareto optimal solutions, or a predefined number of them. One widely used method is the ε-constraint method [80], proposed by Haimes et al. in 1971. This method can be used to produce solutions which can be proven to always be weakly Pareto optimal [42]. The ε-constraint method operates in an iterative way. At each iteration, one of the criteria is optimized and other criteria are constrained with specified values. This procedure can be repeated until finding the entire set of solutions–in case it is a finite set, or until finding a predefined number of solutions. There are several methods in the litertature for systematically modifying the constraints values, see for example [12, 14].

As Approach 2 minimizes power consumption under a constrained latency (deadlines can be replaced by any threshold value), we propose a third approach, denoted

[2]Note that in the literature, *strongly Pareto optimal solutions* are often simply referred to by *Pareto optimal solutions*, see for example [42].

Approach 3, relying on a variant of the ε-constraint method [80] and Approach 2 to generate weak and strong Pareto optimal solutions.

5.4 Evaluation

5.4.1 Experiment 1: Evaluation of Approach 1 and Approach 2

In this Experiment, we first evaluate the run-time of Approach 1 and Approach 2 presented in Section 5.3. In Approach 1, where two phases of DSE (latency optimization followed by power optimization) are performed, the timeout of 1800 seconds applies to the power optimization phase. As Approach 2 is performed in a single phase, the timeout applies to the entire DSE flow. We consider the same set of workloads from Chapters 3 and 4, Figure 3.11, and Architecture A used for evaluations in Chapter 3 and Chapter 4, Figure 3.16.A set of 5 operating modes is available for the bus connecting the CPU (8, 16, 24, 32, and 64 *du*/time slot), and for the two other buses, 4 operating modes are available (8, 16, 24, and 32 *du*/time slot). The bus to which the fifth mode (64 *du*/ time slot) is associated is the bus most in demand because it connects the generic CPU, which communicates with the PUs connected to the other buses. In the rest of this Section, we will refer to Approach 1 by *minlat-minpow* and to Approach 2 by *dea-minpow*, for more expressive notations.

Results are given in Figure 5.1. We can see that Approach 2 (*dea-minpow*) requires longer run-times to be solved compared to Approach 1 (*minlat-minpow*), which is expected. There are several reasons that can explain this: In fact, since minimal latencies (*minlat-minpow*) are shorter than deadlines (*dea-minpow*), the time interval over which power consumption of buses is evaluated is smaller, and so is the number of time slots. Moreover, the search space guaranteeing minimum latencies is smaller compared to the case where only deadlines are respected, as minimizing latencies limits the mapping and scheduling choices to those that can guarantee the minimum latencies (unless deadlines are equal to minimum latencies). The solver wasn't able to finish the exploration within the timeout for 2 out of 15 workloads for *minlat-minpow*, and 8 out of 15 workloads for *dea-minpow*.

The latency optimization phase which results are given in Chapter 4 for Architecture A (Figure 4.9), completed in less than 100 seconds for all workloads. In fact, the

Figure 5.1: Run-time to search the minimal power consumption: 1) under deadlines constraints (*dea-minpow*), 2) under minimum latencies (*minlat-minpow*), on Architecture A, as a function of different workloads.

power optimization phase is more difficult since the power consumption should be evaluated for each possible amount of data that each bus can transfer (ranging from 0 to the maximum capacity of the operating mode), on each slot, with respect to all available operating modes.

In terms of power consumption minimization, from Table 5.1, it can be seen that *minlat-minpow* reduces power consumption of the interconnect by 23.02% on average, for our testbench, compared to the solution obtained after the latency optimization phase. For *dea-minpow*, the average reduction of power consumption on our testbench is measured to 40.12%. The gap between the percentage of power reduction in *dea-minpow* solutions (40.12% on average) and *minlat-minpow* solutions (23.02% on average) shows that solutions which only have to meet deadlines offer

Table 5.1: Interconnect power reduction percentage and approximate overall power reduction for all workloads, on Architecture A, for *dea-minpow* and *minlat-minpow*.

Workload	Reduction(%) dea-minpow		Reduction(%) minlat-minpow	
	Interconnect power (%)	Overall power(%)	Interconnect power (%)	Overall power(%)
sobel	57.75%	1.43%	28.64%	0.71%
susan	21.05%	0.07%	0.00%	0.00%
rasta	21.05%	0.47%	0.00%	0.00%
sora	53.52%	1.25%	25.34%	0.58%
sosu	55.94%	0.48%	27.23%	0.23%
sura	21.05%	0.20%	25.00%	0.24%
sosura	52.67%	0.68%	24.68%	0.32%
jpeg	33.68%	0.27%	28.64%	0.11%
sojp	46.48%	0.30%	28.64%	0.18%
sujp	44.94%	0.17%	26.69%	0.10%
sosujp	36.09%	0.28%	27.75%	0.15%
sorajp	35.97%	0.43%	28.71%	0.23%
rajp	34.98%	0.37%	24.43%	0.17%
surajp	42.54%	0.29%	23.72%	0.14%
sosurajp	44.16%	0.33%	25.83%	0.17%
Average	40.12%	0.47%	23.02%	0.22%

greater room for improvement on power than solutions that have to minimize latency. On the same table, we calculate an approximation of the percentage of power reduction of each approach with respect to the overall power consumption (processing and communication). Calculated values show that the reduced power consumption represents only 0.47% (dea-minpow) and 0.22% (minlat-minpow) of the overall power consumption. Although these values are very low for the studied testbench, we want to emphasize that the focus of this chapter lies in the modeling of power consumption of the multi-bus interconnect, and the study of the capability of the DSE to account for more than one design objective. As part of our future work, we plan to extend the power module of this chapter with a power module for Processing Units (PUs) to study the overall power consumption and explore whether more significant reductions could be obtained.

5.4.2 Experiment 2: Generating Pareto optimal solutions using Approach 3

In this experiment, we study the generation of Pareto optimal solutions to produce solutions with diverse trade-offs in terms of latency and power consumption. For this, we consider application *RASTA-PLP* (workload *rasta*) running on Architecture A (Figure 3.16).

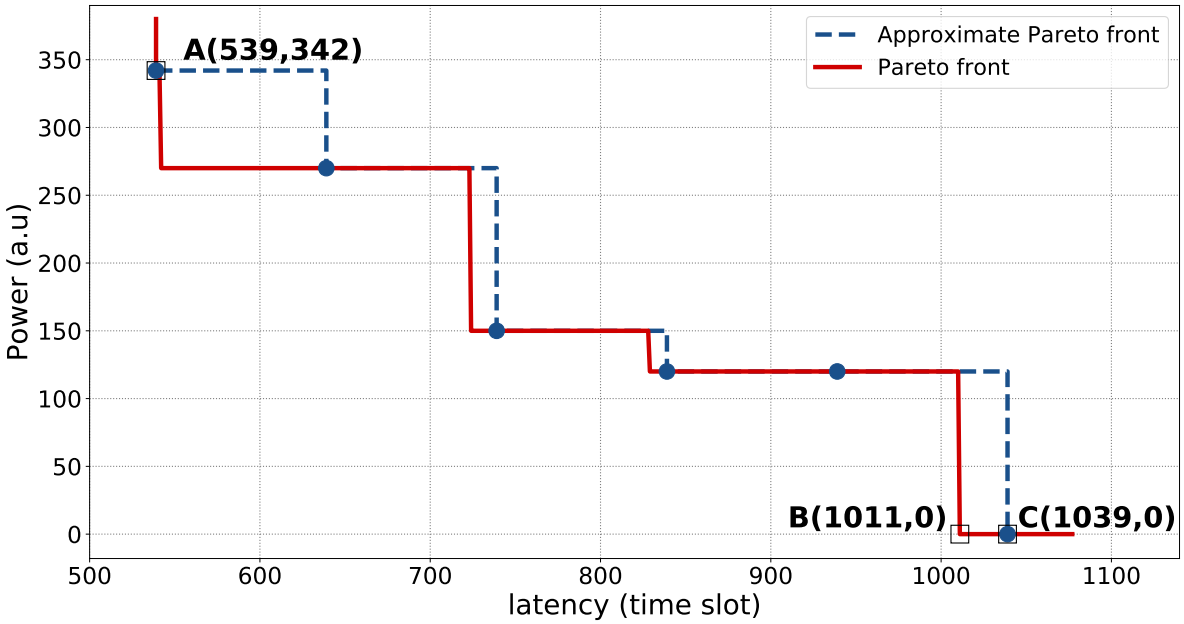


Figure 5.2: (Blue dashed staircase) Approximate Pareto front and (solid red staircase) actual Pareto front found using Approach 3.

We perform a first experiment that aims at generating a set of Pareto optimal solutions. For this, we start the search with the solution obtained using minlat-minpow. By construction, this first solution (marked by point A in Figure 5.2) is a strong Pareto optimum, and more precisely it is the Pareto optimum having the smallest possible latency. From this point, we use the so-called regular grid method [1] to compute a subset of the Pareto front. We retrieve the latency of the solution obtained by minlat-minpow and increase it by an arbitrary amount of 100 time slots, and we use the resulting value to constrain dea-minpow. For instance, in Figure 5.2 the latency of the first solution is 539 time slots, thus yielding a new constraint of 639 time slots.

With this constrain, dea-minpow gives a new solution, which is also by construction a Pareto optimum, and the latency of which is at most 639 time slots. We proceed to find a third Pareto Optimum, and so on and so forth, until some chosen maximal latency value. Several remarks are in order:

- The chosen value for the latency increment defines the granularity of the Pareto front obtained by our method. The smaller this value, the finer the granularity, i.e., the more points we have in the obtained Pareto front.
- Except the first point, obtained with minlat-minpow, none of the subsequent points are guaranteed to be strong Pareto optima. But they are guaranteed to be weak Pareto optima thanks to our usage of an SMT solver.
- Because of the granularity imposed by the latency increment, the Pareto front that we obtain is a subset of the exact Pareto front. In Figure 5.2, the exact Pareto front is depicted by the solid red staircase, while the Pareto front obtained with the grid method is depicted by the dashed blue staircase. The granularity is also a reason why the points are not guaranteed to be strong Pareto optima (except the first one).
- Finally, the chosen value for the maximal latency value is also a reason why we produce a subset of the exact Pareto front.

The search can be stopped after a number of predefined solutions are found, or when a timeout is reached. Here, we stopped the search when we reached zero power consumption (point C), which corresponds to solutions where the interconnect is not used because all tasks are mapped to the same processing unit. From the dashed blue staircase, we can see that the trade-off between power consumption and latency is correctly captured: power consumption increases when latency decreases and vice versa. We can also see that the 6 solutions found provide diverse trade-offs between latency and power consumption. Points that tend to be at the right end of the blue dashed staircase reduce power at the expense of a higher latency while points at the left end reduce latency at the expense of a higher power cost. Points in the middle of the staircase provide a more balanced trade-off in terms of latency and power consumption. The decision is left to the designer to select the

most appropriate design from his/her perspective. The approximate Pareto front can be found within the timeout of 30 minutes (precisely in 9 minutes and 37 seconds for the studied case). The largest part of the total search time is dedicated to finding solutions under larger threshold values. From right to left, finding a Pareto optimal solution on the blue dashed staircase took respectively 64.49% (point C), 28.93%, 4.61%, 1.46%, 0.48% and 0.03% (point A) of the total search time. This unbalance is caused by the variability of the extent of freedom degrees allowed (e.g. mapping choices) below each different threshold on latency.

The solid red staircase in Figure 5.2 depicts the actual Pareto front. It can be obtained by covering all values that the threshold on latency can take (here all integer values ranging from 539 to 1039 as going beyond 1039 will only produce solutions with higher latency without any gain on power). However, the cost of the exhaustive search is significantly higher than the cost of finding the approximate Pareto front. It took 15 hours and 43 minutes to search for all Pareto optimal solutions which is much higher than the timeout we fixed.

From Figure 5.2, we can see that there are both weakly and strongly Pareto optimal solutions. For example, point C is as good as point B in terms of power consumption but point B offers a reduced latency compared to point C. Point C is a weakly Pareto optimal solution, and point B is a strongly Pareto optimal solution. The same trend can be seen all over the solid red staircase, where horizontal segments represent sets of solutions with the same power consumption but different latencies. From a design point of view, on each horizontal segment of the solid red staircase, unless there are other criteria to consider, the solution at the left end (strong Pareto optimum) is the best since it is as good as all other solutions on the same segment (weakly Pareto optimal solutions) in terms of power consumption but has the lowest latency.

As an alternative to searching exhaustively Pareto optimal solutions, some optimizations of the search process can be proposed to guide the generation of good quality solutions. For instance, we can search in the neighborhood of each of the weakly Pareto optimal solutions initially found (blue points), if there are other solutions that are strictly better in latency and no worse in power consumption. For this, for each of the solutions depicted in blue points (except point A which is already strongly Pareto optimal), we can minimize latency under the constraint of power con-

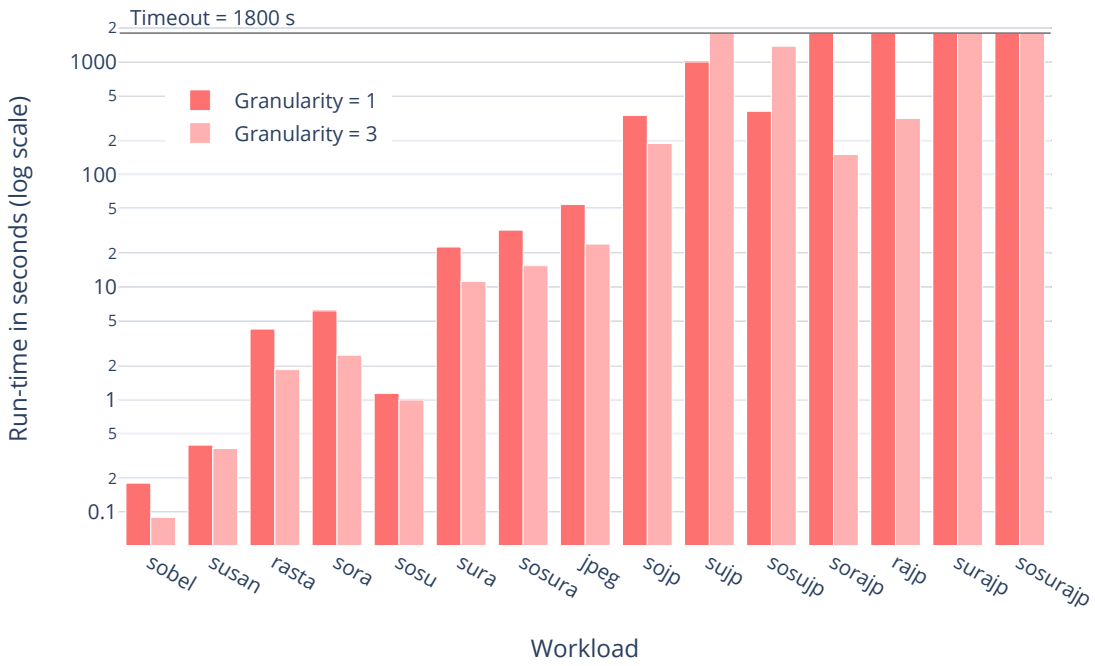


Figure 5.3: Run-time to search the minimal power consumption under minimum latencies on Architecture B, using two levels of granularity (1 and 3), as a function of different workloads.

sumption initially found. Thus, strongly Pareto optimal solutions can be generated.

5.4.3 Experiment 3: Evaluating granularity effect

This Experiment is similar to Experiment 4, Chapter 4: we evaluate the effect of the slot granularity on the run-time of the power optimization phase and investigate whether granularity allows to reduce run-time and solve larger scale problems within a fixed timeout. For this experiment, we will focus on Approach 1.

We use the same set of architectures from Experiment 4, Chapter 4, Figures 4.10, 4.11, 4.12 and 4.13, and the set of all workloads previously used. A set of 4 operating modes is available for all buses, in all architectures, where bandwidth capacity in each mode is equal to 8, 16, 24, and 32 *du*/ time slot. The granularity was set to: 1)

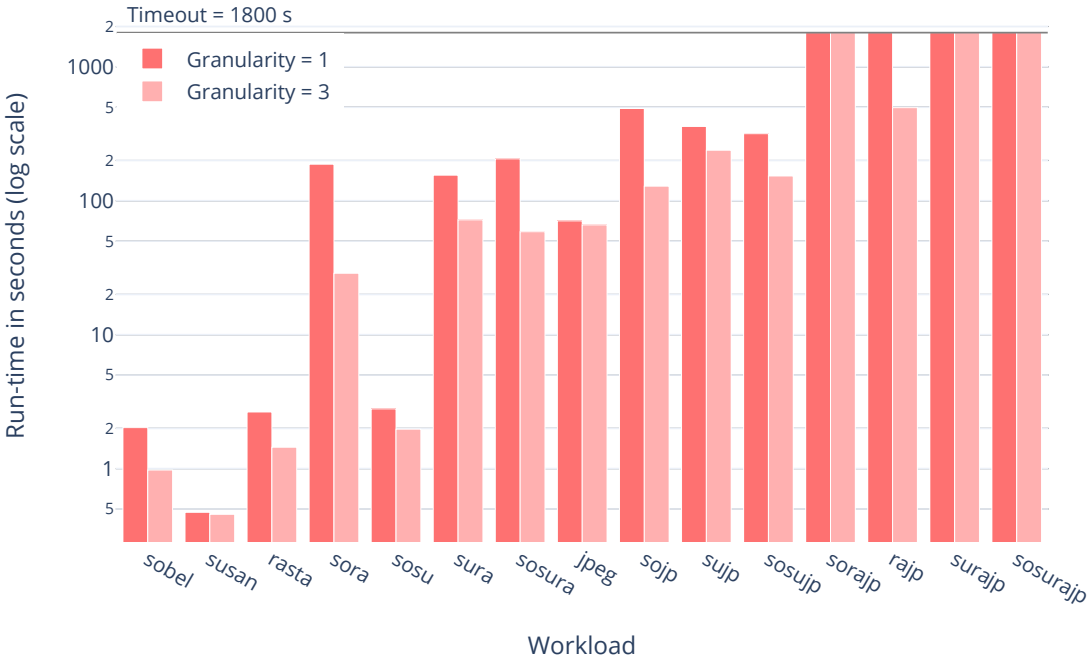


Figure 5.4: Run-time to search the minimal power consumption under minimum latencies on Architecture C, using two levels of granularity (1 and 3), as a function of different workloads.

$granularity = 1$, i.e., $1slot = 1\ cycle$ and 2) $granularity = 3$, i.e., $1slot = 3\ cycle$.

Results are collected in Figures 5.3, 5.4, 5.5 and 5.6 and in Table 5.2. Figures 5.3- 5.6 show the evolution of run-time of the power minimization phase of Approach 1, for all workloads, on different architectures. It can be seen that in most cases (44 out of 60), a coarser granularity reduces run-time. The speed-up and slow-down factors are given in Table 5.2, where positive values represent the speed-up achieved with a $granularity = 3$ compared to a $granularity = 1$, and negative values represent the slow-down observed for some instances with a $granularity = 3$ compared to a $granularity = 1$. On average, a positive speed-up of 23.05 can be calculated from the table. For 5 out of the total 60 tested instances, the coarser granularity unexpectedly slows-down the resolution (e.g., *sosujp*, Table 5.2a). We were not able to identify the precise reason for this result but it can be seen that the coarser

Table 5.2: Quality gap (%) and speed-up (positive values) or slow-down (negative values) factor of solutions found at granularity = 3, compared to solutions found at granularity = 1, for a set of workloads.

workload	gap (%)	speed-up / slow-down
sobel	1.41	2.01
susan	17.62	1.07
rasta	17.62	2.29
sora	2.91	2.48
sosu	0.82	1.14
sura	17.62	2.02
sosura	3.48	2.07
jpeg	1.41	2.24
sojp	1.41	1.77
sujp	0.36	$\leq$-1,82
sosujp	0.29	-3.85
sorajp	1.23	$\geq$11.98
rajp	2.29	$\geq$5.73
surajp	2.87	timeout
sosurajp	1.76	timeout

(a) Architecture B.

workload	gap	speed-up / slow-down
sobel	1.41	2.07
susan	17.62	1.04
rasta	17.62	1.83
sora	2.29	6.52
sosu	1.41	1.42
sura	17.62	2.15
sosura	1.74	3.50
jpeg	1.41	1.07
sojp	1.41	3.80
sujp	0.36	1.50
sosujp	1.41	2.07
sorajp	0.98	timeout
rajp	3.58	$\geq$3.63
surajp	4.12	timeout
sosurajp	0.74	timeout

(b) Architecture C.

workload	gap (%)	speed-up / slow-down
sobel	1.41	2.20
susan	17.62	-1,19
rasta	17.62	51.85
sora	2.91	-2,04
sosu	0.82	1.78
sura	17.62	13.55
sosura	3.48	14.94
jpeg	1.41	2.37
sojp	1.41	1.84
sujp	0.36	$\leq$-1,33
sosujp	0.29	$\geq$2.41
sorajp	1.23	$\geq$1.13
rajp	2.29	$\geq$1.22
surajp	2.87	timeout
sosurajp	1.76	$\geq$2.27

(c) Architecture D.

workload	gap (%)	speed-up / slow-down
sobel	1.41	2.24
susan	17.62	-1,01
rasta	17.62	100.92
sora	2.91	85.50
sosu	1.85	8.09
sura	16.97	3.29
sosura	3.93	2.58
jpeg	1.41	3.90
sojp	1.41	2.51
sujp	1.82	timeout
sosujp	2.23	$\geq$5.22
sorajp	1.23	timeout
rajp	2.29	$\geq$1.48
surajp	2.72	timeout
sosurajp	2.40	timeout

(d) Architecture E.

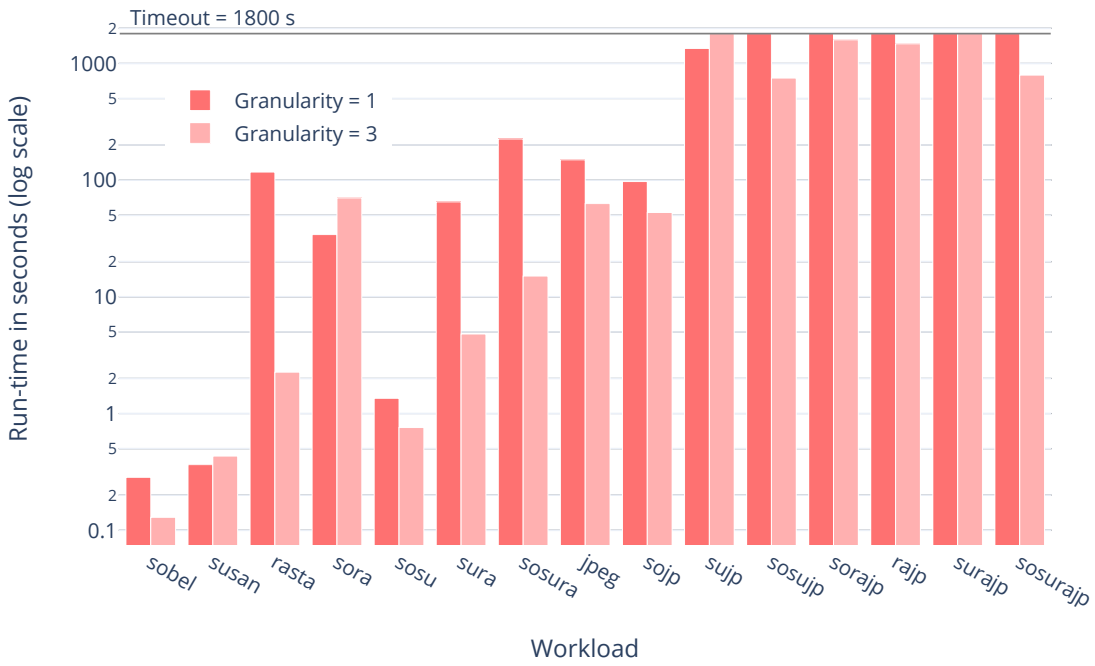


Figure 5.5: Run-time to search the minimal power consumption under minimum latencies on Architecture D, using two levels of granularity (1 and 3), as a function of different workloads.

granularity can make the problem harder to solve in some situations. For 10 out of 60 instances, the effect of the coarser granularity could not be evaluated as the exploration was not able to complete within the timeout with both granularity levels. For *sosurajp*, Table 5.2c, the speed-up is at least equal to 2.27 because the exploration at $granularity = 1$ didn't complete within the timeout, thereby, the exact speed-up could not be calculated but it is at least equal to 2.27. The same explanation applies to other instances with a "$\geq$" sign. For instances with a "$\leq$" sign, the slow-down factor is lower than the value given, since the exploration at $granularity = 3$ didn't complete within the timeout, while it did at $granularity = 1$. For example, *sujp* in Table 5.2c is slowed-down at least by a factor of 1.33.

Overall, at $granularity = 3$, 48 out of the 60 instances studied were solved within the fixed timeout, while at $granularity = 1$, only 41 out of 60 instances were solved.

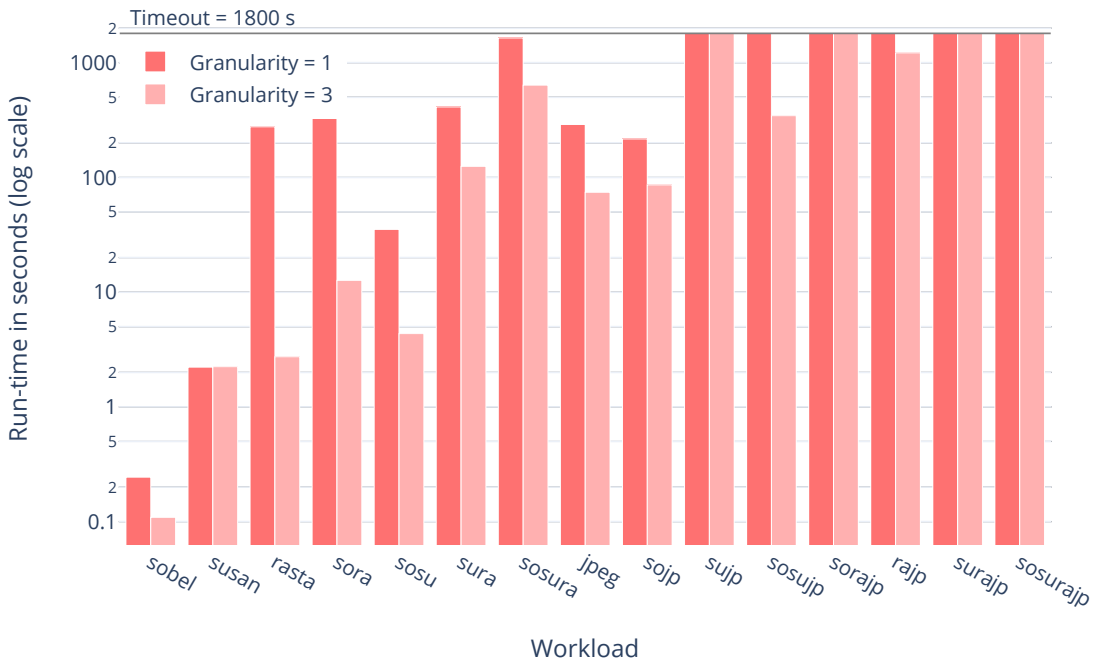


Figure 5.6: Run-time to search the minimal power consumption under minimum latencies on Architecture E, using two levels of granularity (1 and 3), as a function of different workloads.

The quality gap in terms of power consumption between the solutions—optimal or best solution found within the timeout—found at $granularity = 3$ and $granularity = 1$ is given in Table 5.2c. The average gap can be calculated to 4.95%.

Finally, regardless of the granularity level, we can see that Architecture E, Figure 5.6 has higher run-times than Architecture B, Figure 5.3, while the same number of buses and processing resources are considered. This indicates that more routing options, present in the ring topology of Architecture E, increase the DSE run-time.

5.5 Conclusion

In this Chapter, we first presented a SMT formulation for the power consumption of a multi-bus communication architecture. This formulation captures power consumed

by each bus segment under different operating modes. Second, we studied how it can extend the latency model of previous Chapters to address the latency-power consumption bi-criteria problem.

We have shown that by using the same SMT modeling, different bi-criteria optimization approaches can be encoded such that satisfaction levels among the two conflicting objectives are either articulated before the exploration, or after a set of solutions have been produced.

Three approaches were proposed: 1) minimize power consumption under minimal latencies, 2) minimize power consumption under deadlines constraints on latencies, and 3) generate a set of Pareto optimal solutions. Results showed that either a single or multiple (weak and/or strong) Pareto optimal solutions could be generated using the proposed approaches. As a matter of fact, different trade-offs between latency and power consumption can be studied. A common limitation to the proposed approaches is their high run-time. Therefore, we studied the effect of variable slot granularity on the power optimization phase of Approach 1. Results showed a mixed pattern: for 44 out of the 60 studied workloads, a coarser granularity brings a positive speed-up. However, the resolutions of 5 out of the 60 studied workloads were slowed-down compared to the finer grain slot. For 10 workloads, we were not able to assess the impact of granularity as the timeout is exceeded either way. The average optimality gap is measured to 4.95%. Optimality gaps reported also showed an irregular pattern varying from 0.29% upto 17.62%, which may represent a real limitation to the coarser granularity in power consumption-related estimations.

As part of future work, mainly two limitations of this work could be addressed. First, techniques to reduce run-time for the power optimization phase should be investigated. Once the scalability limitation is relaxed, the power formulation can be extended with additional modules to capture power consumption related to Processing Units (PUs) and to leakage current.

Chapter 6

Conclusion and Future Work

6.1 Summary of the contributions

In this thesis, we proposed a SMT formulation for the DSE of tasks and communications on architectures with a multi-bus interconnect (first contribution). To mitigate the computational costs of the exact search, we proposed a pre-analysis technique (second contribution) that reduces the number of variables, the extent of definition domains, and the number of constraints created by the formulation, without impacting the completeness of the approach. We extended our formulation with a power model to evaluate the interconnect power consumption and study the capabilities and limitations of the proposed DSE to address a bi-criteria problem (third contribution). We integrated these contributions in a design environment based on UML/SysML that we used to perform evaluations on a testbench of real-world streaming applications from [49, 50].

6.1.1 Summary of the first contribution

Chapter 3 presented our first contribution, a mathematical SMT formulation to map and schedule both tasks and communications to architectures with a multi-bus interconnect. The formulation tackles multi-bus interconnects without any constraint on their topology (e.g., hierarchical bus, ring bus, ad-hoc topology). The resulting DSE approach can be used to map and schedule tasks and communication on a fixed

architecture. It can also be used to guide the design of new architectures (e.g., number of buses, interconnect topology), by comparing the resulting solutions for various architectures and selecting the architecture that best meets the design objectives. We proposed a case study (Section 3.8.1) to illustrate how the DSE efficiently assists a designer to select best architecture for a given workload with respect to the design objective of minimizing latencies. We also provided an evaluation of the DSE run-time for a set of workloads and a MPSoC target architecture, Section 3.8.2. This experiment showed that for 8 out of 15 workloads, the solver was not able to return even solutions that satisfy deadlines constraints on latency, within the specified time-out of 30 minutes. This scalability issue is the motivation of our second contribution.

6.1.2 Summary of the second contribution

To cope with the practical limitation of the DSE approach presented in Chapter 3, we proposed a second contribution, in Chapter 4, to speed-up the DSE run-time. We proposed to perform a pre-analysis of applications and architecture graphs, as a first step of the DSE prior to the synthesis of the SMT formulation. This pre-analysis aims at identifying temporal boundaries for the execution of tasks and communications, which are used to reduce the size of the design space, and hence, better focus the exploration only on relevant design solutions. The proposed reduction technique preserves the completeness of DSE, as it only removes non-valid design solutions with respect to the temporal boundaries. We evaluated the efficiency of this contribution to improve the scalability of our initial formulation in Section 4.4: First, we evaluated the speed-up of the SMT solver to find deadline-aware solutions, which for our testbench ranged from 20 up to 589 times. Second, we measured the impact of the reductions on the SMT problem size (i.e., variables and constraints) and found out that it allows reducing the problem up to 95% for the testbench under study. Third, we demonstrated that the reductions enabled to analyze the problem more accurately by allowing DSE to be performed at lower granularity levels, for which it was impossible to complete DSE using the initial formulation of Chapter 3. Then, we demonstrated that the reductions allowed to reach optimal solutions for *all* workloads, while, before the reductions, the solver fails even to find deadline-aware solutions for more than half the workloads. Finally, we provided experiments to study

the scalability of the proposed DSE on different target architectures and showed how the practical limitation of our DSE approach is flexible: larger scale problems can be studied by adjusting the slot granularity while an optimality gap as low as 0.25% on average is observed between a granularity level corresponding to $1\ slot = 3\ cycles$ and $1\ slot = 1\ cycle$.

6.1.3 Summary of the third contribution

As in real-world systems, there are often more than one conflicting criteria, in Chapter 5, we studied the capabilities of the proposed DSE to address a bi-criteria optimization problem. To achieve this, we first extended our formulation with a module to capture power consumption of a multi-bus communication architecture under different operating modes.

We have demonstrated the flexibility of the proposed DSE to adopt different approaches depending on the involvement of the designer in the process of solutions search, and according to his/her preference with respect to the two conflicting objectives. We showed that using the same SMT models and solver, different approaches can be used in practice to answer different requirements, e.g., generate a solution which minimizes power consumption but grants the minimum achievable latency, generate multiple solutions with different trade-offs between latency and power consumption so that preferences are not necessarily articulated a priori. A major limitation concerns the computational effort of the solutions search process, though.

6.1.4 Summary of the integration to a MDE design environment

We integrated the three contributions summarized above into an existing MDE design environment for embedded systems. We selected TTool/DIPLODOCUS [2, 3] as it is free, open-source and targets system-level design of embedded systems. We tightly integrated the Z3 SMT solver [19] within TTool/DIPLODOCUS so as to support the following process: UML/SysML-to-SMT transformation, resolution of the SMT problem, and backtracking of results from Z3 to TTool/DIPLODOCUS. Finally, our contribution is now part of a method starting with the capture of application and architecture models with UML/SysML diagrams, continuing with a DSE, and ending with the generation of code for rapid prototyping on the selected target platform.

6.2 Conclusions

This thesis is focused on DSE for the mapping and scheduling of dataflow applications on a multi-bus architecture. We have shown that contributions dealing with communication-aware DSE do not provide methods and tools able to fully tackle multi-bus DSE problems (e.g., routing, joint mapping and scheduling of tasks and communications), at an early stage of the design flow. The work presented in this thesis contributes to filling this gap by proposing a holistic approach to study communications mapping and scheduling along with the mapping and scheduling of tasks on a multi-bus architecture. Thus, it is possible to perform a holistic DSE on various bus-based architectures ranging from a single shared bus to diverse topologies (e.g., tree, ring, chain). Moreover, the reduction technique proposed in this thesis enabled to improve scalability significantly, thus pushing further the practical limits of a joint scheduling of tasks and communications. Since the reduction technique we define is task-based, it is not specific to multi-bus architectures. Thus, its benefits could also be investigated to any other communication architecture. The proposed DSE is subject to several design assumptions. Even if recognized in similar work, some assumptions are still a bit strong, but can surely be relaxed as we will discuss in the following sections.

6.3 Limitations and Improvements

The assumptions made in this thesis limit the applicability of our contributions to systems that are compliant with these assumptions, or systems for which the impact of these assumptions is acceptable at a high level of abstraction. However, as discussed in Chapter 3, these assumptions (e.g., local memory, negligible intra-PU communication overhead) are recognized in similar work. We discussed in Chapter 3 few workarounds to overcome some of these limitations, namely to account for communications through a shared memory. Of course, another way to enlarge the scope of our work is to directly change the formulation. This is facilitated by the extensibility of the CP-based approach. Another assumption was taken on the presence of architecture components always available to perform data transfers in parallel to processing. The formulation can be changed to map each data transfer to

a dedicated resource (e.g., DMA, PE) from the target architecture during the transfer time.

The proposed DSE is performed at design time to produce a static mapping and scheduling solution. This restricts the applicability of our approach to workloads which are known a priori. Even though static task mapping and scheduling cannot be adapted to dynamic workloads, this is a commonly recognized assumption in DSE literature. A possible way to account for different execution scenarios is to analyze at design time multiple execution scenarios, as in [74]. For example, based on a statistic profiling of the execution of an application under different inputs, multiple execution scenarios could be analyzed leading to different mapping and scheduling solutions.

6.4 Future work

The contributions of this thesis pave the path to many possible future research perspectives. We discuss a few of them in this Section. Mainly, we will discuss two categories of perspectives. The first category, indicated by (spec), concerns extending the formulation with new specifications, i.e., model new aspects of applications, architectures, design constraints and objectives. The second category, indicated by (scale), concerns improving the scalability of the DSE approach.

6.4.1 Extending the approach with new specification

Typically, any new specification that describes new details of application, architecture, or design constraints and objectives can be encoded using the CP modeling. The main challenge to that is the resulting complexity and the applicability to real-world problem sizes. The CP approach, by separating the modeling from the solving of the problem, allows to extend its scope with further specifications, or simplify it to focus on a specific sub-problem (e.g., routing of communications). The contribution of a power module in Chapter 5 is a practical example for this. Other examples are to extend the SMT formulation to account for shared memories as discussed earlier, or to consider reconfigurable blocks (i.e., Field Programmable Gate Array (FPGA)) along with reprogrammable processing resources in order to solve

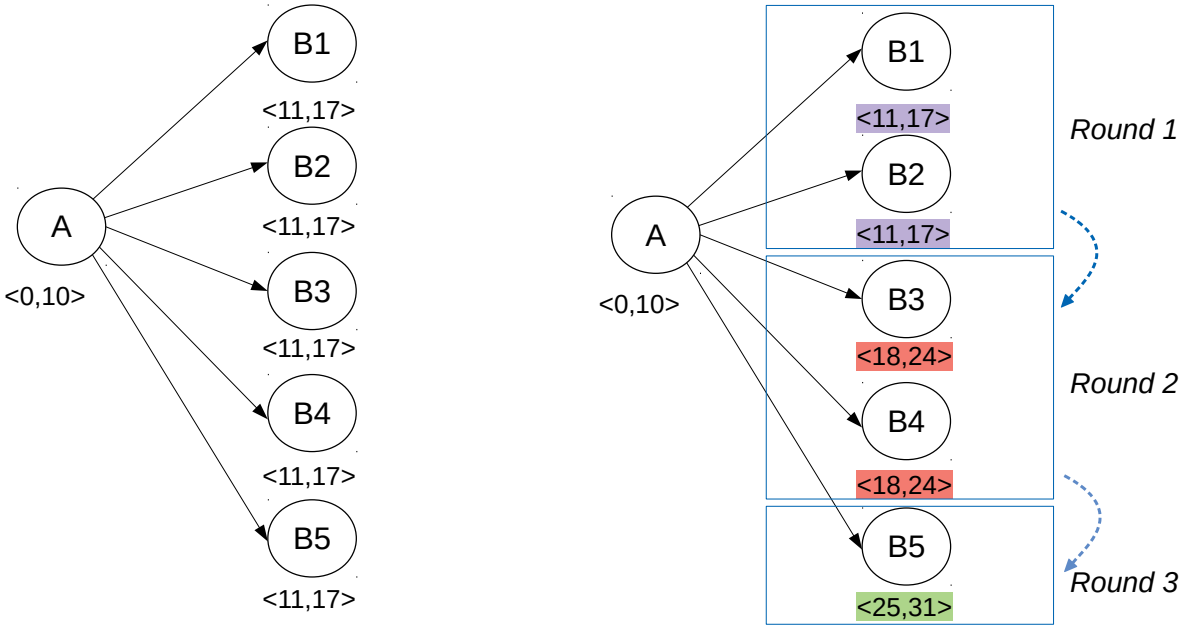
the problem of hardware/software partitioning of tasks. To do this, the formulation can capture FPGA as processing resources that can host as many tasks in parallel as their capacity allows. New input parameters are needed for tasks to denote their requirements in terms of FPGA resource, and for each FPGA to denote its maximum capacity. Mainly, new constraints are required to control that tasks allocated to the same FPGA do not exceed the maximum capacity of this FPGA, i.e. either they shall not run all in parallel, or they shall not be mapped to the same FPGA resource.

Another interesting perspective is to extend the formulation to other types of architectures like NoCs. Here, a promising direction is to investigate the applicability and the benefits of the reduction technique, once NoC-specific behaviors are integrated into the formulation.

6.4.2 Refining the pre-analysis with task mapping information

The pre-analysis for the reduction technique presented in Chapter 4 can be enhanced in order to calculate tighter temporal boundaries, by including more refined task mapping information. More in details, when we calculate the earliest start and finish times of a task, we account for the performance of PEs but not for their availability to execute a given task at a give time. This is illustrated in Figure 6.1a where task A has 5 identical successors $Bi, i \in \{1 \dots 5\}$. This is a typical example of a HSDF obtained from a SDF-to-HSDF graphs transformation, such as in [50]. Here, tasks $Bi, i \in \{1 \dots 5\}$, are identical repetitions of the same SDF actor B according to the SDF theory. This is the case for *JPEG* application, Figure 3.11, where $DCTi, i \in \{1 \dots 6\}$ and $Huffi, i \in \{1 \dots 6\}$ are respectively identical repetitions of SDF actors DCT and $Huff$.

In the example of Figure 6.1a, we assume a target architecture where two PEs are available to execute tasks $Bi, i \in \{1 \dots 5\}$. Hence, the tuples of earliest start and finish times calculated here ($\langle 11, 17 \rangle$), are *actually only valid for two tasks* (e.g., $B1$ and $B2$). Figure 6.1b shows an example of new tuples of earliest start and finish times that could be calculated by the pre-analysis, once the availability of processing resources is taken into account. Here, we can see that, after the completion of task A, two tasks ($B1$ and $B2$) can start executing on the two available PEs. We call it *Round 1*, such that a *Round* is a time stretch on which *all available compatible PEs*



(a) Current pre-analysis.

(b) Enhanced mapping-aware pre-analysis.

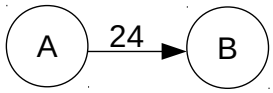
Figure 6.1: Excerpt of an application graph where each task t is annotated with the tuple of earliest start and finish times $\langle ES_t, EF_t \rangle$. Task A has a duration of 11 time slots. Tasks $Bi, i \in \{1 \dots 5\}$ have each a duration of 7 time slots.

are allocated to the tasks. Hence, for remaining tasks, the earliest start and finish times are delayed to subsequent rounds. Three rounds are therefore necessary in our example. In Section 4.4, Chapter 4, we have demonstrated the efficiency of our reductions in reducing the problem size and speeding-up the exploration process. Therefore, such a refinement of temporal boundaries may lead to further significant improvements. This improvement can reduce the impact of the known problem explosion due to SDF-to-HSDF graphs transformation.

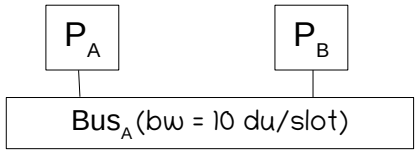
6.4.3 Symmetry breaking in time slots allocation

Symmetry breaking are techniques that aim at identifying the symmetries in the problem—for instance, those present in the architecture graph, and propose proper ways to prevent the search algorithm from analyzing and comparing solutions with identical quality. Since in the context of this work, we mainly focus on heterogeneous architectures, the symmetry breaking for architecture is rather limited. However, a

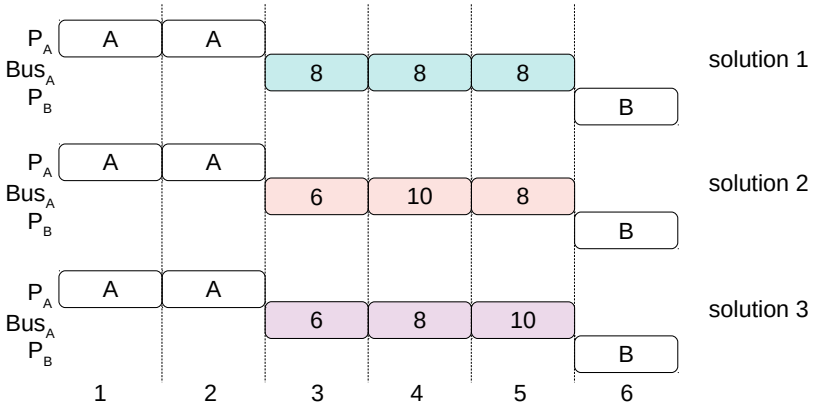
possible direction for future work is to investigate symmetries that can be found in the time slots allocation. Particularly during the latency optimization phase where multiple slot allocation solutions can lead to the same latency. Figure 6.2 shows an example of three scheduling solutions which result into the same quality of latency but where slot allocation is different. Many similar slot allocation solutions can be generated. It would be interesting to investigate how to reduce the number of symmetric solutions, like those illustrated in Figure 6.2c, which are explored and evaluate its impact on DSE run-time. This could also be applied to all situations when we have concurrent communications between different tasks. As long as all these communications complete at the same instant, investigating different communication interleavings is probably not relevant.



(a) Example application excerpt.



(b) Target architecture.



(c) Three possible scheduling solutions where slot allocation, for communication between tasks A and B, are different but result into the same latency equal to 6 time slots.

Figure 6.2: Example of symmetric solutions in slot allocation.

6.4.4 Temporal decomposition

A well-known method to improve scalability of exact search is to decompose the problem into smaller sub-problems. In the literature, decomposition between the problems of mapping and scheduling, or task and communication is widely adopted. However, this traditional decomposition can lead to solutions which are significantly far from the actual optimum. This loss of optimality is due to ignoring inter-dependencies between DSE problems (e.g., mapping, scheduling). Another possible way to leverage the benefit of decomposition is to study the impact of decomposing **temporally** the problem into smaller sub-problems, defined in small time intervals, solve them individually, and merge the results. Here, in contrary to traditional decomposition approaches, on a given time interval, inter-dependencies between mapping and scheduling of tasks and communications are taken into account. For example, in Figure 6.3, every possible combination of mapping and scheduling of tasks t_1, t_2 and t_3 and its impact on the mapping and scheduling of communications between t_1 and t_2 and t_2 and t_3 are analyzed. However, ignoring dependencies between intervals may lead to sub-optimal solutions. For instance, a mapping and scheduling of a communication between two tasks, each included in a different time interval may be sub-optimal because task mapping and scheduling decisions are taken locally for each task. For example, communication between t_1 and t_7 is mapped and scheduled after the mapping of t_1 and t_7 have been decided locally in sub-problems 1 and 3. Therefore, an interesting challenge is to elaborate a methodology to decompose, solve and merge sub-problems. The methodology should define how to cut time intervals and handle inter-intervals communications between tasks such that the loss of optimality is minimal. For example, it can be imagined that after solving individually sub-problems, we reiterate on some portions of the sub-problems which are inter-related, in order to refine the decisions taken locally. This decomposition is likely to improve the scalability and allow enriching the problem with more complex architecture aspects. The challenge remains to find an acceptable trade-off between scalability gain and solution quality loss.

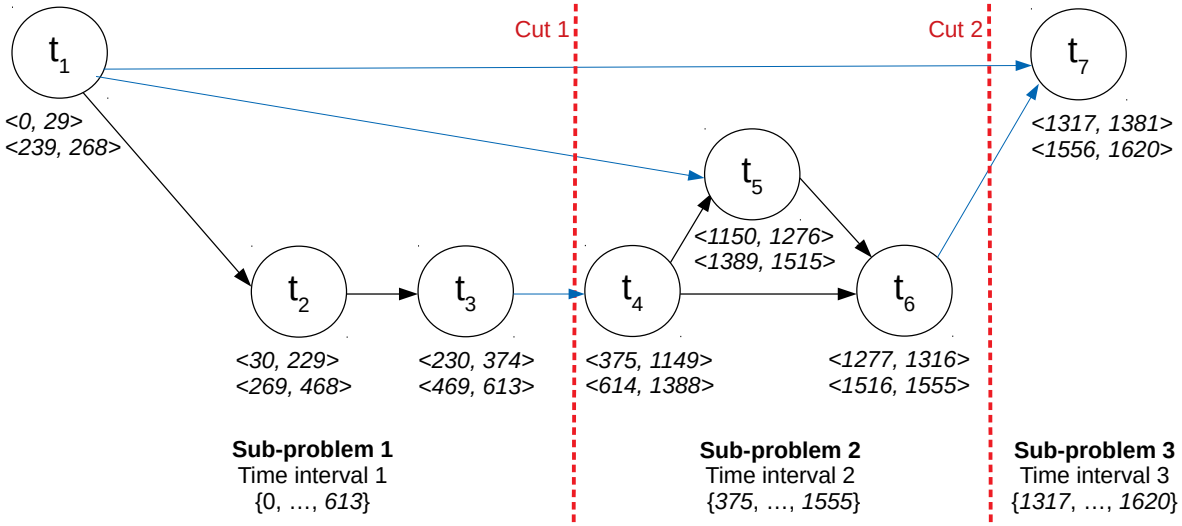


Figure 6.3: Example illustrating time decomposition of the DSE problem.

Bibliography

[1] athena abdi, Alain Girault, and Hamid R. Zarandi. Erpot: A quad-criteria scheduling heuristic to optimize execution time, reliability, power consumption and temperature in multicores. *IEEE Transactions on Parallel and Distributed Systems*, 30(10):2193–2210, 2019.

[2] L. Apvrille, W. Muhammad, R. Ameur-Boulifa, S. Coudert, and R. Pacalet. A UML-based Environment for System Design Space Exploration. In *ICECS*, pages 1272–1275, 2006.

[3] Ludovic Apvrille. Ttool. `http://ttool.telecom-paristech.fr`, 2017.

[4] David Atienza, Federico Angiolini, Srinivasan Murali, Antonio Pullini, Luca Benini, and Giovanni De Micheli. Network-on-chip design and synthesis outlook. *Integration*, 41(3):340–359, 2008.

[5] Brian Bailey, Grant Martin, and Andrew Piziali. Chapter 1 - what is esl? In Brian Bailey, Grant Martin, and Andrew Piziali, editors, *ESL Design and Verification*, Systems on Silicon, pages 1–9. Morgan Kaufmann, Burlington, 2007.

[6] Adarsha Balaji, Yuefeng Wu, Anup Das, Francky Catthoor, and Siebren Schaafsma. Exploration of segmented bus as scalable global interconnect for neuromorphic computing. In *Proceedings of the 2019 on Great Lakes Symposium on VLSI*, GLSVLSI '19, page 495–499, New York, NY, USA, 2019. Association for Computing Machinery.

[7] Luca Benini and Giovanni De Micheli. *Networks on Chips: Technology and Tools*. Systems on Silicon. Morgan Kaufmann, San Francisco, 2006.

[8] Shuvra Bhattacharyya, Praveen Murthy, and Edward Lee. Optimized software synthesis for synchronous dataflow. pages 250–262, 01 1997.

[9] A. Bonfietti, L. Benini, M. Lombardi, and M. Milano. An efficient and complete approach for throughput-maximal sdf allocation and scheduling on multi-core platforms. In *2010 Design, Automation Test in Europe Conference Exhibition (DATE 2010)*, pages 897–902, 2010.

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

[11] Alessio Bonfietti, Michele Lombardi, Michela Milano, and Luca Benini. Maximum-throughput mapping of sdfgs on multi-core soc platforms. *Journal of Parallel and Distributed Computing*, 73(10):1337–1350, 2013.

[12] V. Chankong and Y. Haimes. Multiobjective decision making: Theory and methodology. 1983.

[13] J. Y. Chen, W. B. Jone, J. S. Wang, H. . Lu, and T. F. Chen. Segmented bus design for low-power systems. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 7(1):25–29, 1999.

[14] Kenneth Chircop and David Zammit-Mangion. On epsilon-constraint based methods for the generation of pareto frontiers. *Journal of Mechanics Engineering and Automation*, 3:279–289, 05 2013.

[15] Alessandro Cilardo and Edoardo Fusella. Design automation for application-specific on-chip interconnects: A survey. *Integration*, 52:102–121, 2016.

[16] Alessandro Cilardo, Edoardo Fusella, L. Gallo, A. Mazzeo, and Nicola Mazzocca. Automated design space exploration for fpga-based heterogeneous interconnects. *Design Automation for Embedded Systems*, 18, 03 2014.

[17] Alessandro Cilardo, Edoardo Fusella, Luca Gallo, and Antonino Mazzeo. Exploiting concurrency for the automated synthesis of mpsoc interconnects. *ACM Transactions on Embedded Computing Systems*, 14:1–24, 04 2015.

[18] Indraneel Das and J. E. Dennis. Normal-boundary intersection: A new method for generating the pareto surface in nonlinear multicriteria optimization problems. *SIAM Journal on Optimization*, 8(3):631–657, 1998.

[19] Leonardo de Moura and Nikolaj Bjørner. Z3: An Efficient SMT Solver. In *TACAS*, pages 337–340, 2008.

[20] Leonardo de Moura and Nikolaj Bjørner. Satisfiability modulo theories: An appetizer. In Marcel Vinícius Medeiros Oliveira and Jim Woodcock, editors, *Formal Methods: Foundations and Applications*, pages 23–36, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[21] S. Edwards. "dataflow languages. `http://www.cs.columbia.edu/~sedwards/classes/2001/w4995-02/presentations/dataflow.ppt`, 2001.

[22] M. Fakih, K. Grüttner, M. Fränzle, and A. Rettberg. Towards performance analysis of sdfgs mapped to shared-bus architectures using model-checking. In *2013 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1167–1172, 2013.

[23] FICO. Mathematical programming. `http://www.fico.com/fico-xpress-optimization/docs/dms2018-04/getting_started/dhtml/chap1_sec_c1s1.html`.

[24] Peter C. Fishburn. Lexicographic orders, utilities and decision rules: A survey. 1974.

[25] M. R. Garey and David S. Johnson. Complexity results for multiprocessor scheduling under resource constraints. *SIAM J. Comput.*, 4:397–411, 1975.

[26] Raul Gorcitz, Emilien Kofman, Thomas Carle, Dumitru Potop-Butucaru, and Robert Simone. On the scalability of constraint solving for static/off-line real-time scheduling. In *Formal Modeling and Analysis of Timed Systems*, pages 108–123, Cham, 07 2015. Springer International Publishing.

[27] Matthias Gries. Methods for evaluating and covering the design space during early design development. *Integr. VLSI J.*, 38(2):131–183, December 2004.

[28] Hua Wang, A. Papanikolaou, M. Miranda, and F. Catthoor. A global bus power optimization methodology for physical design of memory dominated systems by coupling bus segmentation and activity driven block placement. In *ASP-DAC 2004: Asia and South Pacific Design Automation Conference 2004 (IEEE Cat. No.04EX753)*, pages 759–761, 2004.

[29] IBM. Mathematical programming vs constraint programming. `http://ibmdecisionoptimization.github.io/docplex-doc/mp_vs_cp.html`.

[30] Ahmed Amine Jerraya and Wayne Wolf. Chapter 1 - the what, why, and how of mpsocs. In Ahmed Amine Jerraya and Wayne Wolf, editors, *Multiprocessor Systems-on-Chips*, Systems on Silicon, pages 1 – 18. Morgan Kaufmann, San Francisco, 2005.

[31] Jui-Ming Chang and M. Pedram. Energy minimization using multiple supply voltages. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 5(4):436–443, 1997.

[32] S. Kamil, L. Oliker, A. Pinar, and J. Shalf. Communication requirements and interconnect optimization for high-end scientific applications. *IEEE Transactions on Parallel and Distributed Systems*, 21(2):188–202, 2010.

[33] James E. Kelley and Morgan R. Walker. Critical-path planning and scheduling. In *IRE-AIEE-ACM*, pages 160–173, 1959.

[34] K. Keutzer, A. R. Newton, J. M. Rabaey, and A. Sangiovanni-Vincentelli. System-level design: orthogonalization of concerns and platform-based design. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 19(12):1523–1543, 2000.

[35] N. Khalilzad, K. Rosvall, and I. Sander. A modular design space exploration framework for multiprocessor real-time systems. In *FDL*, pages 1–7, 2016.

[36] Bart Kienhuis, Ed F. Deprettere, Pieter van der Wolf, and Kees A. Vissers. A methodology to design programmable embedded systems - the y-chart approach. In *Embedded Processor Design Challenges: Systems, Architectures, Modeling, and Simulation - SAMOS*, page 18–37, Berlin, Heidelberg, 2002. Springer-Verlag.

[37] Edward Ashford Lee and David G. Messerschmitt. Static scheduling of synchronous data flow programs for digital signal processing. *IEEE Trans. Comput.*, 36(1):24–35, 1987.

[38] Jing Lin, Andreas Gerstlauer, and Brian L. Evans. Communication-aware heterogeneous multiprocessor mapping for real-time streaming systems. *Journal of Signal Processing Systems*, 69:279–291, 2012.

[39] Mingze Ma and Rizos Sakellariou. Communication-aware scheduling algorithms for synchronous dataflow graphs on multicore systems. In *SAMOS*, pages 55–64, 2018.

[40] R. Marler and Jasbir Arora. Survey of multi-objective optimization methods for engineering. *Structural and Multidisciplinary Optimization*, 26:369–395, 04 2004.

[41] Peter Marwedel. Embedded and cyber-physical systems in a nutshell. *dac.com Knowledge Center Article*, 01 2010.

[42] Kaisa Miettinen. *Introduction to Multiobjective Optimization: Noninteractive Approaches*, pages 1–26. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[43] O. Moreira, J. . Mol, M. Bekooij, and J. van Meerbergen. Multiprocessor resource allocation for hard-real-time streaming with a dynamic job-mix. In *11th IEEE Real Time and Embedded Technology and Applications Symposium*, pages 332–341, 2005.

[44] T. Mudge. Power: a first-class architectural design constraint. *Computer*, 34(4):52–58, 2001.

[45] J. D. Owens, W. J. Dally, R. Ho, D. N. Jayasimha, S. W. Keckler, and L. Peh. Research challenges for on-chip interconnection networks. *IEEE Micro*, 27(5):96–108, 2007.

[46] Sudeep Pasricha and Nikil Dutt. *On-Chip Communication Architectures: System on Chip Interconnect*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2008.

[47] A. D. Pimentel. Exploring exploration: A tutorial introduction to embedded systems design space exploration. *IEEE Design Test*, 34(1):77–90, Feb 2017.

[48] Francesca Rossi, Peter van Beek, and Toby Walsh. *Handbook of Constraint Programming*. Elsevier Science Inc., USA, 2006.

[49] K. Rosvall, T. Mohammadat, G. Ungureanu, J. Öberg, and I. Sander. Exploring Power and Throughput for Dataflow Applications on Predictable NoC Multiprocessors. In *DSD*, pages 719–726, 2018.

[50] K. Rosvall and I. Sander. A constraint-based design space exploration framework for real-time applications on MPSoCs. In *DATE*, pages 1–6, 2014.

[51] Kathrin Rosvall and Ingo Sander. Flexible and Tradeoff-Aware Constraint-Based Design Space Exploration for Streaming Applications on Heterogeneous Platforms. *ACM TODAES*, 23(2):21:1–21:26, 2017.

[52] E Salminen, A Kulmala, and Timo Hämäläinen. Survey of networks-on-chip proposals. *OCP International Partnership*, page 13 p, 01 2009.

[53] A. Sangiovanni-Vincentelli and G. Martin. Platform-based design and software design methodology for embedded systems. *IEEE Design Test of Computers*, 18(6):23–33, 2001.

[54] T. Seceleanu. Communication on a segmented bus. In *IEEE International SOC Conference, 2004. Proceedings.*, pages 205–208, 2004.

[55] T. Seceleanu, V. Leppänen, and O. S. Nevalainen. Improving the performance of bus platforms bymeans of segmentation and optimized resource allocation. *EURASIP J. Embedded Syst.*, 2009, January 2009.

[56] T. Seceleanu, V. Leppanen, J. Suomi, and O. Nevalainen. Resource allocation methodology for the segmented bus platform. In *Proceedings 2005 IEEE International SOC Conference*, pages 129–132, 2005.

[57] T. Seceleanu and S. Stancescu. Arbitration for the segmented bus architecture. In *2004 International Semiconductor Conference. CAS 2004 Proceedings (IEEE Cat. No.04TH8748)*, volume 2, pages 487–490 vol.2, 2004.

[58] Tiberiu Seceleanu. The segbus platform – architecture and communication mechanisms. *Journal of Systems Architecture*, 53(4):151–169, 2007.

[59] A. K. Singh, M. Shafique, A. Kumar, and J. Henkel. Mapping on multi/many-core systems: Survey of current and emerging trends. In *2013 50th ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–10, 2013.

[60] Stephen M. Smith and J. Michael Brady. Susan - a new approach to low level image processing. *International Journal of Computer Vision*, 23(1):45–78, 1997.

[61] Sundararajan Sriram and Shuvra Bhattacharyya. *Embedded Multiprocessors: Scheduling and Synchronization, Second Edition*. 01 2009.

[62] S. Stuijk, T. Basten, M. C. W. Geilen, and H. Corporaal. Multiprocessor resource allocation for throughput-constrained synchronous dataflow graphs. In *2007 44th ACM/IEEE Design Automation Conference*, pages 777–782, 2007.

[63] S. Stuijk, M. Geilen, and T. Basten. Sdf^3: Sdf for free. In *Sixth International Conference on Application of Concurrency to System Design (ACSD'06)*, pages 276–278, 2006.

[64] D. Sylvester and Chenming Wu. Analytical modeling and characterization of deep-submicrometer interconnect. *Proceedings of the IEEE*, 89(5):634–664, 2001.

[65] FRONTLINE SYSTEMS. SOLVER TUTORIAL - SIZE, SPARSITY AND INTEGER VARIABLES. `https://www.solver.com/size-sparsity-integer`.

[66] P. Tendulkar, P. Poplavko, I. Galanommatis, and O. Maler. Many-core scheduling of data parallel applications using smt solvers. In *2014 17th Euromicro Conference on Digital System Design*, pages 615–622, 2014.

[67] Pranav Tendulkar. *Mapping and Scheduling on Multi-core Processors using SMT Solvers*. PhD thesis, University of Grenoble, 2014.

[68] Pranav Tendulkar, Peter Poplavko, and Oded Maler. Symmetry breaking for multi-criteria mapping and scheduling on multicores. In Víctor Braberman and Laurent Fribourg, editors, *Formal Modeling and Analysis of Timed Systems*, pages 228–242, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[69] Kalray. Kalray *MPPA-256*. `http://www.kalray.eu/`.

[70] International Technology Roadmap for Semiconductors (ITRS) The ITRS Technology Working Groups. `http://www.public.itrs.net`, 2005.

[71] William Thies. *Language and Compiler Support for Stream Programs*. PhD thesis, USA, 2009. AAI0821753.

[72] TTool. `http://ttool.telecom-paristech.fr/diplodocus.html`, 2006.

[73] Erik B. van der Tol and Egbert G.T. Jaspers. Mapping of MPEG-4 decoding on a flexible architecture platform. In Sethuraman Panchanathan, V. Michael Bove Jr., and Subramania I. Sudharsanan, editors, *Media Processors 2002*, volume 4674, pages 1 – 13. International Society for Optics and Photonics, SPIE, 2001.

[74] Peter van Stralen and Andy Pimentel. Scenario-based design space exploration of mpsocs. In *2010 IEEE International Conference on Computer Design*, pages 305–312, 2010.

[75] S. Voss and B. Schätz. Deployment and scheduling synthesis for mixed-critical shared-memory applications. In *2013 20th IEEE International Conference and Workshops on Engineering of Computer Based Systems (ECBS)*, pages 100–109, 2013.

[76] Sebastian Voss, Johannes Eder, and Florian Hölzl. Design Space Exploration and its Visualization in AUTOFOCUS3. In *SE Workshop*, volume 1129, pages 57–66, 2014.

[77] P. Wielage and K. Goossens. Networks on silicon: blessing or nightmare? In *Proceedings Euromicro Symposium on Digital System Design. Architectures, Methods and Tools*, pages 196–200, 2002.

[78] W. Wolf. The future of multiprocessor systems-on-chips. In *Proceedings. 41st Design Automation Conference, 2004.*, pages 681–685, 2004.

[79] W. Wolf, A. A. Jerraya, and G. Martin. Multiprocessor system-on-chip (mpsoc) technology. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 27(10):1701–1713, 2008.

[80] Yv Haimes Yv, L. Lasdon, and Dang Da. On a bicriterion formation of the problems of integrated system identification and system optimization. *IEEE Transactions on Systems, Man, and Cybernetics*, pages 296–297, 1971.

[81] Y. Zhang, R. Y. Chen, W. Ye, and M. J. Irwin. System level interconnect power modeling. In *Proceedings Eleventh Annual IEEE International ASIC Conference (Cat. No.98TH8372)*, pages 289–293, 1998.

Titre: Une approche basée programmation par contraintes pour l'exploration d'architectures multi bus pour les applications flots de données

Mots clés: Exploration d'architecture, Programmation par contraintes, Satisfiability Modulo Theories, UML / SysML, Ingénierie des modèles, Flot de données

Résumé: Cette thèse a été effectuée à Télécom Paris et a été financée par Nokia Bell Labs France. Dans ce contexte, nous nous intéressons à l'exploration d'architecture des systèmes embarqués pour le déploiement des applications de traitement de signal, au niveau système. Ici, l'exploration d'architecture vise à identifier l'allocation et l'ordonnancement des deux composants des applications : les tâches et leurs transferts des données. Cette identification a un impact clé sur la performance (e.g., latence de bout en bout) globale du système. Tandis que plusieurs travaux se sont intéressés aux diverses architectures de communication, cette thèse se focalise sur les architectures multi-bus, particulièrement adaptées aux plateformes de calcul pour les applications de traitement de signal. Pour ce type de plateformes, nous montrons que les contributions déjà proposées sont insuffisantes. A cet égard, nous proposons trois contributions : 1) Une formulation satisfiability modulo theories (SMT) qui permet d'explorer les décisions d'allocation et d'ordonnancement sur les architectures multi-bus pour l'optimisation de la latence ; Nous démontrons son applicabilité pour produire des solutions pour des applications connues. 2) Pour améliorer la scalabilité de la recherche optimale de la première contribution, nous proposons une nouvelle technique pour couper l'espace des solutions recherchées. Notre évaluation démontre un gain de scalabilité. Finalement, 3) la consommation de puissance par les communications est étudiée ; nous montrons comment optimiser la latence et la consommation conjointement. Nos évaluations montrent comment différents compromis entre latence et consommation de puissance peuvent être étudiés. De plus, nous montrons comment nos contribution ont été intégrées à un outil de modélisation et de vérification particulièrement adapté à la conception des systèmes embarqués au niveau système (TTool). Enfin, nous identifions deux axes principaux pour les perspectives de ce travail. Le premier porte sur l'extension de la formulation actuelle pour modéliser de nouveaux aspects des systèmes étudiés (e.g., mémoire partagée, débit). Le deuxième axe concerne l'élaboration de nouvelles techniques pour améliorer davantage la scalabilité de la recherche optimale.

Title: Constraint Programming for Design Space Exploration of Dataflow Applications on Multi-Bus Architectures

Keywords: Design Space Exploration, Constraint Programming, Satisfiability Modulo Theories, UML / SysML, Data Flow, Model-Driven Engineering

Abstract: This thesis is part of a collaboration between Télécom Paris and Nokia Bell Labs France. In this context, we focus on the system-level Design Space Exploration of embedded systems for the execution of signal processing applications. In the system we target, the design space exploration process intends to identify the allocation and scheduling of both application tasks and data transfers between these tasks: this identification plays a key role in the overall performance (e.g., end-to-end latency) of these systems. While there are already multiple work for diverse communication architectures, this thesis focuses on multi-bus architectures that are particularly well-suited for computation platforms of signal processing applications. For these platforms, we show that only limited contributions have already been proposed. Three contributions are proposed to tackle the above mentioned problem. 1) A satisfiability modulo theories (SMT) formulation which allows to explore mapping and scheduling decisions on multi-bus architectures for latency optimization; We demonstrate its ability to produce a solution for well-known applications. Yet, 2) to mitigate the scalability limitations for the optimal solution search of this first contribution, we propose a technique to prune the design space of searched solutions. Evaluations we provide demonstrate a better scalability. Last, 3) communication allocation is enhanced with power consumption, and we show how to jointly optimize latency and power consumption. Our evaluation is again applied to a set of well-known signal processing applications and demonstrates how different trade-offs between latency and power consumption can be studied. Our contributions are integrated into a state-of-the-art modeling and verification tool for the system-level design of embedded systems (TTool). Perspectives are articulated in mainly two axes. 1) Extending the current formulation to account for new design aspects (e.g., shared memory, throughput). 2) Further improving the scalability of the optimal search.

Institut Polytechnique de Paris
91120 Palaiseau, France