# Deployment of loop-intensive applications on heterogeneous multiprocessor architectures

Philippe Anicet Glanon

université
PARIS-SACLAY

# Deployment of Loop-intensive applications on Heterogeneous Multiprocessor Architectures

**Thèse de doctorat de l'Université Paris-Saclay**

École doctorale n° 580, Sciences et Technologies de l'Information et de la Communication (STIC)
Spécialité de doctorat: Programmation, Modèles, Algorithmes, Architecture, Réseaux
Unité de recherche: CEA-LIST, 8 Avenue de la vauve, 91120 Palaiseau
Référent: Faculté des Sciences d'Orsay

**Thèse présentée et soutenue au CEA LIST en Amphi 33/34, le 16 Octobre 2020 à 10h00, par**

**Philippe GLANON**

**Composition du jury:**

| | |
|---|---|
| **Christine Eisenbeis**<br>Directrice de recherche, INRIA Paris-Saclay | Présidente & examinatrice |
| **Laurent Pautet**<br>Professeur, Télécom Paris | Rapporteur |
| **Maxime Pelcat**<br>Associate Professor, INSA Rennes | Rapporteur |
| **Alix Munier-Kordon**<br>Professeure, Sorbonne Université | Examinatrice |
| **Chokri Mraidha**<br>Ingénieur-Chercheur, HDR, CEA LIST | Directeur de thèse |
| **Selma Azaiez**<br>Ingénieur-Chercheur, PhD, CEA LIST | Co-encadrante |

Thèse de doctorat

NNT: 2020UPASG029

# *Remerciements*

Avant tout, je tiens à remercier mon directeur de thèse, monsieur Chokri Mraidha et mon encadrante, madame Selma Azaiez. Je me suis senti chanceux d'avoir mené pendant trois ans, mes travaux de thèse sous leur direction au CEA LIST. Ils ont cru en moi dès le début de ma thèse et m'ont constamment motivé et encouragé pendant ces trois années. Je les remercie plus particulièrement pour leurs encadrements de qualité, leurs disponibilités, leur spontanéité et pour leur professionnalisme remarquable qui m'ont permis de bien mener mes travaux de thèse dans le temps imparti.

J'adresse ma profonde gratitude à messieurs Christian Gamrat, Etienne Hamelin, Olivier Heron et madame Marie-Isabelle Giudici pour m'avoir accueilli au CEA LIST au sein du laboratoire LCYL (ex L3S) et mis à ma disposition des moyens financiers, logistiques et techniques pour bien mener mes travaux de thèse.

Je voudrais ensuite remercier messieurs Laurent Pautet et Maxime Pelcat pour avoir accepté d'examiner ce manuscrit de thèse et d'en être les rapporteurs. Leurs différents rapports ont été d'une grande utilité pour l'amélioration du contenu de ce manuscrit. J'étends mes remerciements à mesdames Christine Einsenbeis et Alix Munier-Kordon pour l'attention qu'elles ont portée à mes travaux en acceptant d'être membres du jury.

J'aimerais également remercier toutes les personnes que j'ai rencontrées pendant mes trois années de thèse dans le département DSCIN (ex DACLE) du CEA LIST. En particulier, les ingénieurs-chercheurs P. Dubrulle, M. Asavoe, P. Aubry, S. Carpov, F. Galéa, R. Sirdey, K. Trabelsi, D. Irofti, M. Jan, B. Ben Hedia, Y. Mouafo et P. Ruf avec qui j'ai eu des échanges scientifiques et techniques édifiants. J'adresse aussi mes remerciements à tous mes anciens collègues docteurs et doctorants du DSCIN. En particulier, B. Le Nabec, M. Ait Aba, F. Hebbache, E. Laarouchi, D. Vert, M. Zuber, B. Binder, G. Bettonte, E. Lenormand et A. Madi pour toutes les discussions intéressantes que nous avions eu aux pauses cafés.

J'adresse un grand merci à tous mes proches et amis pour leur soutien inconditionnel. Je voudrais spécialement remercier Joannes, José, Ingrid, Jeannot, Fawaz, Auriol, Catherine, Valentin, Juliette et Christo qui n'ont jamais cessé de me soutenir et d'égayer mes journées.

Enfin, je voudrais remercier ma grand-mère Simone, mes parents Marlène et Innocent, ma soeur Mitzrael et ma dulcinée Sandrine pour tout l'amour, le soutien et les soins qu'ils me portent au quotidien.

# Résumé

Les systèmes cyber-physiques sont des systèmes distribués qui intègrent un large panel d'applications logicielles et de ressources de calcul hétérogènes connectées par divers moyens de communication (filaire ou non-filaire). Ces systèmes ont pour caractéristique de traiter en temps-réel, un volume important de données provenant de processus physiques, chimiques ou biologiques. Une des problématiques rencontrée dans la phase de conception des systèmes cyber-physiques est de prédire le comportement temporel des applications logicielles. Afin de répondre à cette problématique, des stratégies d'ordonnancement statique sont nécessaires. Ces stratégies doivent tenir compte de plusieurs contraintes, notamment les contraintes de dépendances cycliques induites par l'exécution des boucles de calculs spécifiées dans les programmes logiciels ainsi que les contraintes de ressource et de communication inhérentes aux architectures matérielles de calcul. En effet, les boucles étant l'une des parties les plus critiques en temps d'exécution pour plusieurs applications de calcul intensif, le comportement temporel et les performances optimales des applications logicielles dépendent de l'ordonnancement optimal des structures de boucles spécifiées dans les programmes de calcul. Pour prédire le comportement temporel des applications logicielles et fournir des garanties de performances dans la phase de conception au plus tôt, les stratégies d'ordonnancement statiques doivent explorer et exploiter efficacement le parallélisme embarqué dans les patterns d'exécution des programmes à boucles intensives tout en garantissant le respect des contraintes de ressources et de communication des architectures de calcul.

L'ordonnancement d'un programme à boucles intensives sous contraintes ressources et communication est un problème complexe et difficile. Afin de résoudre efficacement ce problème, il est indispensable de concevoir des heuristiques. Cependant, pour concevoir des heuristiques efficaces, il est important de caractériser l'ensemble des solutions optimales pour le problème d'ordonnancement. Une solution optimale pour un problème d'ordonnancement est un ordonnancement qui réalise un objectif optimal de performance.

Dans cette thèse, nous adressons le problème d'ordonnancement des programmes à boucles intensives sur des architectures de calcul multiprocesseurs hétérogènes sous des contraintes de ressource et de communication, avec l'objectif d'optimiser le débit de fonctionnement des applications logicielles. Pour ce faire, nous nous sommes focalisés sur l'utilisation des graphes de flots de données synchrones. Ces graphes sont des modèles de calculs permettant de décrire les structures de boucles spécifiées dans les programmes logiciels de calcul et d'explorer le parallélisme embarqué dans ces structures de boucles à travers des stratégies d'ordonnancement cyclique. Un graphe de flot de donnée synchrone est un graphe orienté constitué d'un nombre fini de noeuds appelés "acteurs" et d'un nombre fini d'arcs appelés "canaux". Un noeud représente une tâche de calcul dans un programme logiciel et un arc est une file d'attente "premier arrivé, premier servi" qui modélise le flux de données échangées entre deux acteurs. Chaque arc est constitué d'un taux de production, d'un taux de consommation et d'un marquage initial potentiellement nul. Les taux de production et de consommation d'un arc représentent respectivement la quantité de données produite par l'acteur en aval de l'arc et la quantité de données consommée par l'acteur en amont de l'arc. Le marquage initial quant à lui décrit la quantité de donnée initialement présente sur un arc. Lorsqu'un arc contient un marquage initial non-nul, il induit des relations de dépendances inter-itération entre les diverses instances d'exécution des acteurs connectés.

Dans la première partie de cette thèse, nous montrons en quoi l'utilisation des graphes de flots de données synchrones est bénéfique pour la modélisation des structures de boucles spécifiées dans les programmes logiciels des systèmes cyber-physiques. Dans la deuxième partie, nous proposons des stratégies d'ordonnancement cycliques basés sur la propriété mathématiques des graphes de flots de données synchrones pour générer des solutions optimales et approximatives d'ordonnancement sous les contraintes de ressources et de communication des architectures de calcul multiprocesseurs hétérogènes.

# Mots-clés

système cyber-physiques, ordonnancement multiprocesseur, graphes de flots de données statiques, architectures hétérogènes, pipeline logiciel, débit maximal.

# abstract

Cyber-physical systems (CPSs) are distributed computing-intensive systems, that integrate a wide range of software applications and heterogeneous processing resources, each interacting with the other ones through different communication resources to process a large volume of data sensed from physical, chemical or biological processes. An essential issue in the design stage of these systems is to predict the timing behaviour of software applications and to provide performance guarantee to these applications. In order tackle this issue, efficient static scheduling strategies are required to deploy the computations of software applications on the processing architectures. These scheduling strategies should deal with several constraints, which include the loop-carried dependency constraints between the computational programs as well as the resource and communication constraints of the processing architectures intended to execute these programs. Actually, loops being one of the most time-critical parts of many computing-intensive applications, the optimal timing behaviour and performance of the applications depends on the optimal schedule of loops structures enclosed in the computational programs executed by the applications. Therefore, to provide performance guarantee for the applications, the scheduling strategies should efficiently explore and exploit the parallelism embedded in the repetitive execution patterns of loops while ensuring the respect of resource and communications constraints of the processing architectures of CPSs. Scheduling a loop under resource and communication constraints is a complex problem. To solve it efficiently, heuristics are obviously necessary. However, to design efficient heuristics, it is important to characterize the set of optimal solutions for the scheduling problem. An optimal solution for a scheduling problem is a schedule that achieve an optimal performance goal. In this thesis, we tackle the study of resource-constrained and communication-constrained scheduling of loop-intensive applications on heterogeneous multiprocessor architectures with the goal of optimizing throughput performance for the applications. In order to characterize the set of optimal scheduling solutions and to design efficient scheduling heuristics, we use synchronous dataflow (SDF)

model of computation to describe the loop structures specified in the computational programs of software applications and we design software pipelined scheduling strategies based on the structural and mathematical properties of the SDF model.


## Keywords

cyber-physical systems; multiprocessor scheduling; static dataflow graphs; heterogeneous architectures; software pipelining; maximum throughput

# Table of Contents

# List of Figures

# List of Tables

# List of Symbols

- $\mathbb{Z}$: the set of integers.
- $\mathbb{N}$ the set of positive integers.
- $\mathbb{Q}^{+}$: the set of positive rationals.
- $\mathbb{Q}^{+*}$: the set of strictly positive rationals.

# CHAPTER 1

# Introduction

## Contents

## 1.1 General Context and Problem Statement

During the past decades, advances in hardware and software technologies have led to the development of modern computing systems called cyber-physical systems (CPSs). These systems are attracting a lot of attention in recent years and are being considered as innovative technologies that can improve human life and address many technical, socio-economical and environmental challenges.

**What are CPSs?** There is a plethora of definitions in the literature [2, 4, 6, 7, 8, 9] that may agree with our vision. We believe that CPSs are integrations of distributed computing components which interact with each other through wired or wireless communication resources to sense and control physical processes. Unlike traditional embedded systems (smartphones, digital watches, etc.), which are designed as standalone devices, the focus in a CPS is on networking several devices or sub-systems and commanding them remotely in order to interact with physical processes. CPSs find applications in a wide range of domains including manufacturing, healthcare, environment, transportation. In the manufacturing industry, they are being introduced to characterize the upcoming of the fourth industrial revolution, frequently noted as Industry 4.0 [2, 4, 5, 6]. A manufacturing CPS (also called cyber-physical production system) is a production system where equipment such as robots, automated guided vehicles (AGVs), sensors and controllers interact with each other to control and monitor manufacturing operations at all levels of the production, from physical transformation processes through machines up to logistics network. In the healthcare sector, CPSs are being used to remotely monitor the health conditions of in-hospital and in-home patients and to provide healthcare services [7]. A healthcare CPS is a system that collects the health data of patients through various medical sensors, transmits these data to a gateway via a wired or a wireless communication medium, stores the data in a cloud server and makes these data accessible to caregivers. In environment and transportation areas, large scale CPSs named smart cities [3] are being deployed to improve the service efficiency and quality of life in the cities. A smart city is one including digitally connected service systems such as transportation, power distribution and public safety systems, which are integrated using various information and communication technologies. In such a city, we will travel in driverless cars that communicate with each other on smart roads and in planes that coordinate to reduce delays. Homes and offices will be powered by a smart grid that use sensors to analyze the environment and optimize energy in real time.

**Design Requirements.** Each of the CPSs previously presented consists of multiple software applications and embedded computing platforms. The software applications are es-

sentially loop-intensive applications —i.e. applications that perform repetitive computations —that generate a huge volume of data processed by the computing platforms. These platforms are usually multiprocessor systems that include a finite number of heterogeneous processing resources, each communicating with the other ones through different communication means. Thus, the computations of software applications can be easily parallelized by distributing data across different processing resources. An important design requirement of CPSs is that the processing resources of a computing platform may be shared between the computations of one or more applications. This requirement can lead to resource conflicts and/or communication bottlenecks when different computations need to access the same resources at the same time, and thus, it can cause a loss of parallelism and a noticeable deterioration of performance achievable by the software applications. To prevent such a situation, CPS designers often need to explore the parallelization choices of computations to different types of multiprocessor architectures. For this purpose, the design approaches for CPSs should be based as much as possible on formal models of computation that deal both with time, concurrency and parallelism. These models should be implementable and analyzable so that the designers may use them to predict the timing behaviour of CPS applications and to provide performance guarantees for these applications at design stage. Among the popular models of computation, dataflow models are of high interest.

**Dataflow models.** Dataflow modeling paradigm is characterized by a data-driven style of control where data are processed while flowing through a networks of computation nodes. A dataflow model is a directed graph where nodes (called actors) describe the computations performed by a loop-intensive application and edges are FIFO channels that describe the dependency relations between the computations. When an actor fires, it consumes a finite number of data tokens and produces a finite number of data tokens. A set of firing rules indicates when the actor is enabled to fire. Dataflow models are often classified into dynamic and static models. Dynamic dataflow models are known to be more expressive than static dataflow models. However, the Turing-completeness and non-decidability of these models has motivated the research community to adopt static dataflow models when it comes to implement and analyze the behaviour of an application. Various types of static dataflow models exists. One of the most known is synchronous dataflow (SDF) model. In a SDF graph, the number of tokens consumed and produced by each actor at each firing is predefined at design stage. SDF graphs has been traditionally used to design streaming and multimedia applications. There interests are increasingly growing nowadays in the CPS design because of their semantics that enables to describe different levels of parallelism in loop-intensive applications and to analyze the timing behaviour and performance of these

applications through the construction of static-order schedules (i.e. infinite repetitions of firing sequences of actors) with bounded FIFO channels.

**Scheduling.** Scheduling a static dataflow graph consists in finding "when" the firings of each actor must be executed. An optimal schedule of a static graph is a schedule that achieves an optimal performance goal. Interesting performance indicators often analyzed to evaluate the optimality of schedules for static dataflow graphs are usually throughput and latency. From a CPS perspective, the study of these metrics is important to predict the timing behaviour of CPS applications and to provide performance guarantees for these applications. Scheduling strategies for static dataflow graphs can be classified into self-timed schedules (also called as soon as possible schedules) and periodic schedules. In a self-timed schedule, the instances of actors are executed as soon as possible the required data are available while in a periodic schedule, the instances of actors are executed according to a fix time period. Self-timed schedules are known as scheduling strategies that achieve optimal throughput for static dataflow graphs. However, these schedules are more difficult to implement than periodic schedules. A common way to get around the implementation complexity of self-timed schedules is to implement software pipelined (SWP) schedules [33, 60]. SWP schedules are a subclass of periodic schedules widely used to analyze the timing behaviour of loop-intensive applications. These schedules provide the same guarantees than self-timed schedules in terms of throughput achievement.

**Problem statement.** In this thesis, we address the following problem: *given a CPS application modeled by a SDF graph and a CPS platform described by a heterogeneous multiprocessor architecture with a fixed number of communicating processing resources, how can one construct an optimal SWP schedule that achieves the highest performance of the SDF graph under the resource and communication constraints of the given architecture?* Scheduling an application graph under resource and/or communication constraints is a NP-hard problem [29, 59]. Therefore, the problem tackled by this thesis is NP-hard. The main objective of the thesis is to propose efficient strategies to solve this scheduling problem.

## 1.2 Contributions

In order to solve the problem stated above, we have made several contributions to the scheduling of SDF graphs. Our main contributions are the following ones:

- First, we characterize the set of admissible SWP schedules that achieve optimal throughput/latency for timed SDF graphs and we propose linear programming mod-

els to compute these schedules.

- Secondly, we show that the problem can be model as an integer linear programming (ILP) model with precise optimization constraints and objective. The ILP model accommodates pipeline, task and data parallelism in SDF graphs and it characterizes the set of SWP scheduling solutions that achieves optimal throughput.

- Thirdly, we propose a guaranteed decomposed SWP scheduling heuristic, that generates approximated SWP scheduling solutions for the problem.

## 1.3 Thesis Organization

This thesis is organized into two parts:

- **Part 1.** This part presents the motivations, state-of-the-art and a detailed formulation of the problem tackled by this thesis. The part consists of two chapters. The first chapter (chapter 2) gives a quick overview of design requirements and tools for cyber-physical systems. In this chapter, we show why heterogeneous multiprocessor architectures and synchronous dataflow (SDF) model are suited to the design of cyber-physical systems and we present the motivations that pushed us to be interested in the scheduling of SDF graphs on multiprocessor architectures under resources and communication constraints. In the second chapter (chapter 3), we review the basics of the SDF model, we formulate the main problem tackled by this thesis and we present some related works.

- **Part 2.** In this part we present our contributions. The part is organized into four chapters. In the first chapter (chapter 4), we propose a theorem that characterizes the set of admissible SWP schedules for timed SDF graphs. In this chapter, we also present two linear programming models, one enabling to compute SWP schedules that achieve maximum throughput for timed SDF graphs and the other to compute SWP schedules that achieve minimum latency. In the second chapter (chapter 5), we extend the characterizations made in chapter 4 to formulate an ILP model used to generate SWP scheduling solutions that achieve maximum throughput for SDF graphs on heterogeneous multiprocessor architectures under resource and communication constraints. In this chapter, we also present our decomposed SWP scheduling heuristic that generates approximated scheduling solutions for the resource- constrained and communication-constrained scheduling problem. In the third chapter

(chapter 6), we validate of our contributions thanks to experimental results made with synthetic benchmarks and real-world application benchmarks. In the fourth chapter (chapter 7), we summarize our contributions and present some open research challenges.

# Part I

# Motivations, State-of-the-Art and Problem Formulation

# CHAPTER 2

# Background and Motivations

## Contents

## 2.1 Introduction

This chapter presents the background and motivations of this thesis. In the chapter, we give a quick overview of cyber-physical systems, the multiprocessor architectures, the parallel programming models and we present the motivations that pushed us to be interested in the scheduling of SDF graphs on heterogeneous multiprocessor architectures under resources and communication constraints.

## 2.2 Architecture of Cyber-Physical Systems

Cyber-physical systems (CPSs) can generally be represented as a control-loop structure like that depicted in figure 2.1. There are three parts in this structure. Firstly, there is a **physical plant**, which represents the "physical part" of a CPS. This part can include human operators, mechanical parts, biological or chemical processes. Secondly, there are computing **platforms**, each with its own sensors, computing features and/or actuators. Thirdly, there is a **network fabric**, which provides the mechanisms for the platforms to communicate. Together, the platforms and the network fabric constitute the "cyber part" of a CPS. Figure 2.1 illustrates a CPS structure composed with two computing platforms.

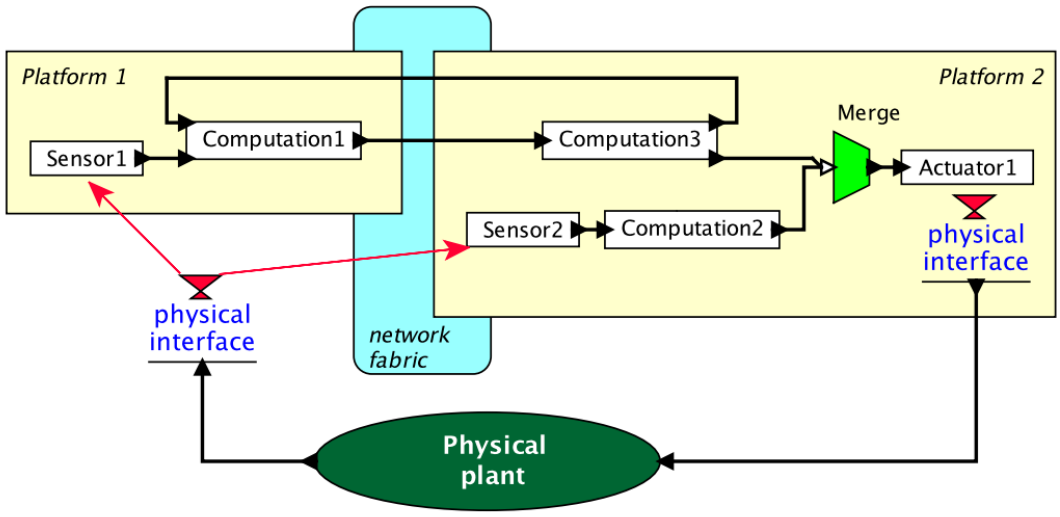
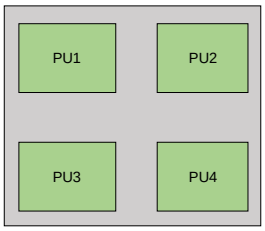
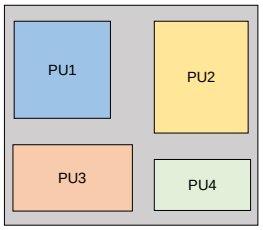



Figure 2.1: Example Structure of a CPS (Lee & Seshia [8])

Platform 2 measures the processes in the physical plant using sensor 2 and it controls the physical plant via actuator 1. The box labelled "Computation 2" represents a control application which uses the data collected by sensor 2 data to generate the command laws that trigger the behaviour of the actuator."Computation 3" realizes an additional control




(a) Homogeneous architecture




(b) Heterogeneous architecture

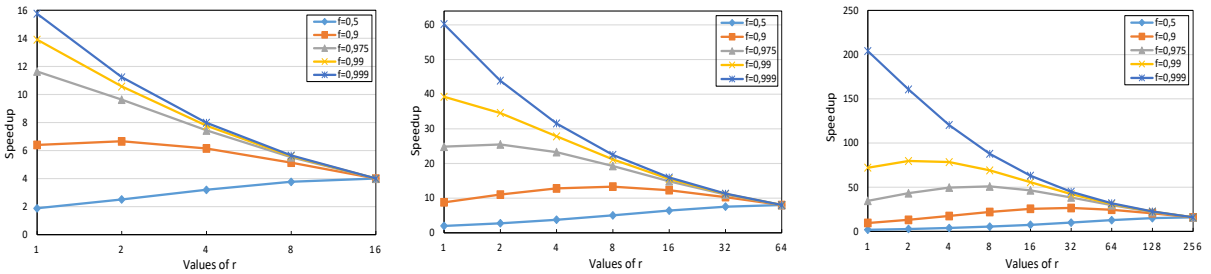
Figure 2.2: Illustration of different types of multiprocessor architectures

law, which is merged with that of "Computation 2" and Platform 1 makes additional measurements via sensor 1, and sends messages to Platform 2 via the network fabric.

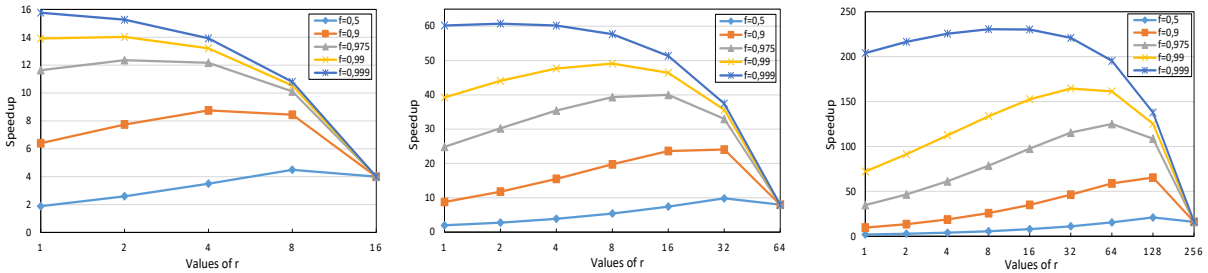
A common requirement of a CPS is the need of processing a large volume of data collected by various sensors in order to control the physical plant at real-time. For this purpose, multiprocessor computing architectures are often implemented to parallelize the computations of CPS. Over the past decades, different types of multiprocessor architectures have been proposed to parallelize computations in many computer systems. These architectures can be classified into homogeneous and heterogeneous architectures. Figure 2.2 gives a graphical illustration of these different types of architectures. A homogeneous multiprocessor architecture includes multiple processing units (PUs) which have the same micro-architecture and/or provide the same computing performance while a heterogeneous architecture combines different types of PUs each having a specific micro-architecture and/or a providing specific computing performance. A systematic question that arises when designing the "cyber" part of a CPS, is whether a homogeneous or heterogeneous architectures should be used. In order to provide an answer to this question, Amdahl's law [13] can be used to compute the performance provided by both types of systems. This law finds the maximum expected improvement of an overall computer system when only a part of the system is improved. Let speedup be the original execution time of a software program divided by an enhanced execution time. The modern version of Amdahl's law states that if a fraction $f$ of a software program is enhanced by a speedup $S$, then the overall speedup of this program is given by:

$$Speedup_{enhanced}(f, S) = \frac{1}{(1-f) + \frac{f}{S}} \tag{2.1}$$

In the context of multicore and multiprocessor systems, the authors in [14] have provided a corollary of Amdahl's law. Let us consider a multiprocessor system (or a multicore




(a) Homogeneous with n=16 (b) Homogeneous with n=64 (c) Homogeneous with n=256




(d) Heterogeneous with n=16 (e) Heterogeneous with n=64 (f) Heterogeneous with n=256

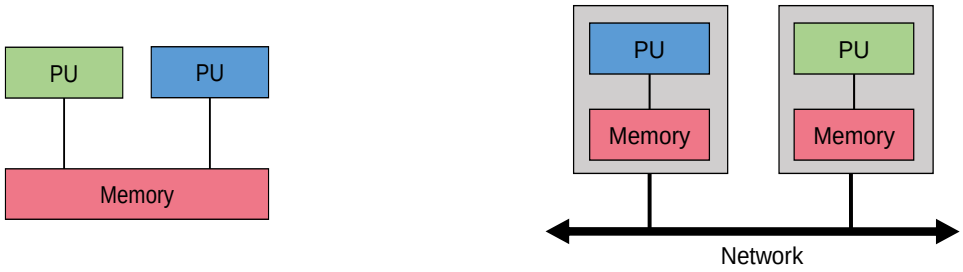
Figure 2.3: Speedup of homogeneous and heterogeneous multiprocessor/multicore systems.

machine) with $n$ processors (or cores). Under Amdahl's law, the overall speedup of such a system depends on the fraction $f$ of software program that can be parallelized, the number $n$ of processors in the system and the number $r$ of base processors that can be combined to build one bigger processor (or core). In the case of a homogeneous multiprocessor system, one processor is used to execute sequentially the software program at performance $perf(r)$ and $n/r$ processors are used to execute in parallel the program at performance *perf (r)*$\times n/r$. Thus, the overall speedup obtained with this system is given by:

$$Speedup_{homogeneous}(f,n,r) = \frac{1}{\frac{1-f}{perf(r)} + \frac{f \cdot r}{perf(r) \cdot n}} \tag{2.2}$$

In the case of a heterogeneous multiprocessor system, only the processor with more computation resources is used to execute sequentially at performance perf(r). In the parallel fraction, however, it gets performance *perf(r)* from the large processor and performance 1 from each of the $n-r$ base processors. Thus, the overall speedup obtained with this system is given by:

$$Speedup_{heterogeneous}(f,n,r) = \frac{1}{\frac{1-f}{perf(r)} + \frac{f}{perf(r)+n-r}} \tag{2.3}$$




(a) Shared memory architecture (b) Distributed memory architecture

Figure 2.4: Multiprocessor architectures according to the memory access criteria

Figure 2.3 plots the speedup obtained for both homogeneous and heterogeneous multiprocessors/multicore systems with respect to different values of the parameters $f$, $n$ and $r$. In these plots, we assume that $perf(f) = \sqrt{r}$ as done in [14]. As it can be seen, for a same number of processing units, the speedups obtained with a heterogeneous system is much better than those obtained with a homogeneous system. This observation is an important reason for using heterogeneous multiprocessor architectures for CPSs design.

## 2.3 Parallel Programming Paradigm

Previously shown, heterogeneous multiprocessor architectures have an attractive advantage to carry out efficiently parallel computations in CPSs. In order to efficiently exploit the potential of these architectures, different hardware programming mechanisms have been proposed. However, their utilization is extremely complex for designers and programmers. To overcome this complexity, many programming approaches focus on the specification of software applications intended to run on these computer architectures. Thus, a plethora of application programming models have been proposed to allow users to write their software programs and to specify parallelism in these programs. These models can be classified into multithreading and dataflow models.

### 2.3.1 Multithreading Programming Models

Multithreading is a programming paradigm based on the utilization of threads to specify concurrency and parallelism in software programs. A thread is a way of making a program to execute two or more computational tasks at the same time. A thread consists of its own program counter, its own stack and a copy of registers of central processing unit (CPUs), but it shares other things like the code that is executing, heap and some data structures. Several multithreading programming models exist. PThread, OpenMP, MPI and OpenCL

are four of the well-known multithreading programming models.

**Pthread.** Pthread also known as POSIX thread [20] is a low-level application programming model for writing concurrent software programs for shared-memory computer architectures (refer to figure 2.4a). It consists of a library of functions used to define threads accessing to a shared-memory space. The programming model is built on the top of imperative programming language like C and it supports different variants of operating systems including Unix, Windows and Mac OS. Although Pthread has its place in specialized situations, the mechanisms for synchronizing threads is not explicitly defined, which makes the utilization of this programming model difficult for programmers to develop correct and maintainable computer programs [16, 15].

**OpenMP.** OpenMP [17] is a high-level programming model to write parallel software programs for shared-memory computer systems ranging from desktops to supercomputers. The implementation of OpenMP exists for three different programming languages including Fortran, C and C++. Contrary to POSIX Thread, the utilization of threads OpenMP is highly structured and threads are implicitly synchronized. In an OpenMP program, when an executing thread encounters a "parallel" directive, it will create a group of threads and become the master thread of this group. Then the group of threads executes the program code assigned to it. When the group has terminated its execution, the master thread collects the results from the group of threads and serially executes from that point on. For example if a "for" loop that will iterate over an array containing 100 elements would be parallelized on a processor with 4 cores, OpenMP will be used to create four threads and execute one thread on each core. The "for" loop will be wrapped in a parallel directive where the limit of threads is four. Then, when the initial thread encounters this directive a fork would occur, four tasks would be created and the goal for each task is to iterate over a subset of the array. This will increase the performance by four times, sometimes more if it benefits from super-linear speedup [17]. Recently, OpenMP has been extended for heterogeneous multiprocessor architectures, which makes it one of the two predominant models for programming many parallel computing systems, the other being Message Passing Interface (MPI) [18].

**MPI.** As opposed to Pthread and OpenMP, MPI [18] is a programming model that was developed for writing portable software programs for distributed-memory computer architectures (refer to figure 2.4b). Similar to OpenMP, the implementations of MPI is also available for C, C++, and Fortran programming languages. Programming with MPI have an attractive advantage that it provides point-to-point and collective communication mod-

els to specify the communicating threads, without having to manage their synchronization. In an MPI program, a computation comprises one or more threads that communicate by calling library routines to send and receive messages to other threads. In most MPI implementations, a fixed set of threads is created at the program initialization, and one thread is created per processor. However, these threads may execute different programs. Hence, the MPI programming model is sometimes referred to as a multiple program multiple data model to distinguish it from the single program multiple data model in which every processing element executes the same program. MPI programming model remains the dominant model used for designing parallel computing application today[19].

**OpenCL.** OpenCL [21] is a low-level standardized programming model designed to support the development of portable software programs intended to run on heterogeneous computing systems with shared memory architectures. OpenCL views a computing system as a set of devices which might be central processing units (CPUs), graphics processing units (GPUs), digital signal processors (DSPs) or field-programmable gate arrays (FPGAs) attached to a host processor (a CPU). OpenCL programs are divided into host and kernel code. In the host program, kernels and memory movements are queued into command queues associated with a device. The kernel language provides features like vector types and additional memory qualifiers. A computation must be mapped to work-groups of work-items that can be executed in parallel on the processing cores of a device. OpenCL programming model is a good option for mapping threads on different processing units. However, it might be a too low-level programming model for many application-level programmers who would be better served using MPI and OpenMP programming models.

In summary, all of the above mentioned programming models won for general purpose parallel computer systems which involve either shared memory or distributed memory communication mechanisms. However, their utilization is inappropriate in the context of CPSs because of the non-determinism of thread-based programs which can lead to communication overheads, high latencies and/or low throughput and thus decrease the performance of many CPS applications.


(a) KPN (b) HSDF (c) SDF (d) CSDF

Figure 2.5: Examples of the most popular dataflow model.

### 2.3.2 Dataflow Programming Models

Dataflow programming is a visual programming paradigm that appeared with Karp and Miller [66] in the middle of 1960s to specify parallelism in computer programs and to analyze the behaviour of these programs. A dataflow program is described as a directed graph where nodes (called actors) represent computations and arcs (called channels) represent the streams of data flowing between the computations. Actors can communicate with each other by exchanging data onto the channels. Such data are seen as discrete streams of tokens usually represented by dots. A channel may be initialized with a fix number of tokens to describe loop-carried dependency between the execution instances of computations. The quantity of tokens over a channel is called marking and an actor can fire whether there is a sufficient quantity of tokens on its input channels. Dataflow programming models provide high-level abstractions for specifying, analyzing and implementing the behaviour of a wide range of software applications. Nowadays, they are attracting a lot of interests in the design of CPS applications [8]. Various types of dataflow models exist. The most popular are Kahn process networks (KPN), homogeneous static dataflow (HSDF), synchronous dataflow (SDF) and cyclo-static Dataflow (CSDF). Figure 2.5 gives an illustration of each of these models.

**KPN.** KPN is dynamic dataflow model proposed in 1974 by Gilles Kahn [63]. A KPN model (figure 2.5a) is composed of actors linked by unbounded FIFO channels, each connecting a unique pair of actors. Each actor has a sequential behaviour and cannot produce tokens of data on more than one channel at a given time. The firing rules of actors is based on blocking reads and non-blocking writes semantics. This means that an actor can produce a token of data on a channel whenever it wants and must wait when it tries to consume a data token from an empty channel until this data arrives on this channel. The non-blocking write semantics ensures that actor can produce tokens continuously on the channels while the blocking read semantics ensures that any execution order of actors

will yield the same histories of tokens on the channels. KPN is a Turing-complete model [56] that has a high expressiveness to provide a fine-grained description of many computer programs which are inherently partitioned into separate tasks communicating via streams of data. However, like many Turing complete models, KPN is not decidable, i.e. it does not allow deadlocks and performance analysis of computer programs at design time.

**HSDF.** This model is introduced in 1968 by Reiter [65]. HSDF (figure 2.5a) is a static dataflow model also known as single-rate dataflow model in the signal processing community or marked graph in the Petri Net community. This model imposes limitations on the KPN model in order to make deadlocks and performance analysis possible for the computer systems at design time. An HSDF model consists of actors connected with bounded First In First Out (FIFO) channels. The execution instances of actors specifies how many tokens are produced and consumed on each channel. The number of tokens consumed and produced by actors is predetermined and specified on the input and output ports of the channels. When an actor fires, it a consumes a constant number of data tokens from input channels and produces the same number of tokens on output channels. HSDF is less expressive than KPN, however, the bounded capacity of FIFO channels and the constant number of consumption and production rates makes this model more analyzable and easier to implement than the KPN model.

**SDF.** This model is presented in 1987 by Lee and Messerschmitt [61]. SDF (figure 2.5c) is a static dataflow model also known as multi-rate dataflow model in the signal processing community or weighted event graph in the Petri Net community. Similar to the HSDF model, it consists of actors connected by bounded FIFO channels. However, it extends the HSDF model by allowing the specification of different production and consumption rates on the channels. Thus, actors can produce and consume tokens at different rates. In a SDF model, when an actor fires, it consumes a fixed number of tokens —which is predetermined in design time —from each of its incoming channels, and it produces a fixed number of tokens on each of its outgoing channels. Figure 2.5c shows an example of SDF graph composed of two actors ($A$ and $B$) linked by a single channel. The production and consumption rates are indicated respectively at the source and the end of the channel. It should be noted SDF models are more expressive and compact than HSDF models.

**CSDF.** This model is a static dataflow graph introduced in 1995 by Bilsen [46]. It can be seen as an extension of the SDF model. In a CSDF model (figure 2.5d), the production and consumption rates associated to the FIFO channels are decomposed into finite sequences (also called phases) which may change periodically between the execution instances of

Table 2.1: Comparison of dataflow models. Notations: excellent (++++), very good (+++), good (++), less good (+)

| Properties \ Models | KPN | HSDF | SDF | CSDF |
|---|---|---|---|---|
| Expressiveness | ++++ | + | ++ | ++ |
| Succintness | + | ++ | +++ | ++++ |
| Implementation efficiency | + | ++++ | +++ | ++ |
| Analysis potential | + | ++++ | +++ | ++ |

actors. Each actor has a finite number of phases and each phase characterize the number of tokens produced or consumed by this actor. For instance, figure 2.5d illustrates a CSDF with two actors ($A$ and $B$) connected by a single channel. Actor A has an unique production phase where it produces a single token onto the channel while Actor $B$ has two consumption phases where it consumes one token in the first phase and two tokens in the second phase. CSDF is not more expressive than SDF but it is more succinct for describing the computer programs of very large parallel computer systems. However, the complexity to analyze CSDF programs is larger than the analysis complexity of a SDF programs with an equal number of actors [38].

To summarize, SDF, HSDF, and CSDF are static dataflow models that impose a restriction on the KPN model to allow static analysis of computer systems at design time. A detailed comparison of all of these dataflow models have been provided by Stuijk in 2007 [38] according to four criteria including expressiveness, succinctness, implementation efficiency and analysis potential. Table 2.1 presents a summarized comparison of these dataflow models. From this table, we can clearly see that, SDF model is the one that provides a good compromise. In this thesis, we will focus on the utilization of SDF graphs to specify parallelism in CPS applications and to analyze the performance achievable by these applications on heterogeneous multiprocessor architectures.

## 2.4 Deployment of Loop-Intensive Applications

A key aspect in the design of a CPS is the software deployment through which the computations of CPS applications are scheduled and mapped on the PUs of heterogeneous multiprocessor architectures. Scheduling a computation consists to find "when" this computation must start its execution while mapping a computation consists in finding "where" the computation should be executed. The scheduling and mapping of computations for

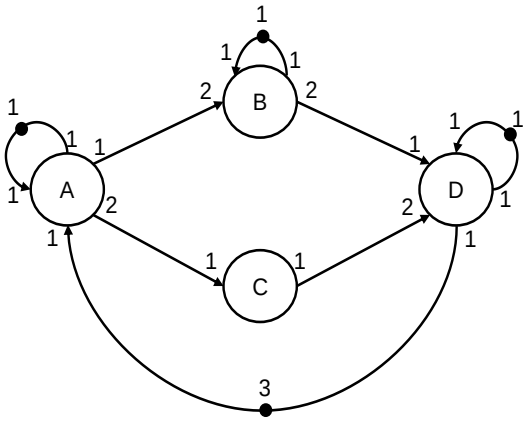
A[k] ← A[k-1] ⊙ D[k-3]

B[k] ← B[k-1] ⊙ A[2k] ⊙ A[2k+1]

C[k] ← A[floor(k/2)]

D[k] ← D[k-1] ⊙ B[floor(k/2)] ⊙ C[2k] ⊙ C [2k+1]

(a) Loop-intensive program




(b) SDF graph

Figure 2.6: A loop-intensive program and its SDF representation

CPS applications depend on many parameters which include for instance, the number of PUs available on the multiprocessor architectures, the worst-case execution times of computations onto the PUs and the inter-PUs communication latencies. In addition to these parameters, loop-carried dependencies should be considered. Actually, loops usually being the most time-critical parts of many software applications, the performance achievable by the applications depends on the optimal execution of loops embedded in the software programs. Therefore, to predict the timing behaviour of CPS applications and to provide performance guarantee at design stage, there is a need of exploring and exploiting the parallelism embedded in the execution pattern of loops. For this purpose, we will show that SDF graphs can be very helpful.

### 2.4.1 Modeling and Exploitation of Parallelism

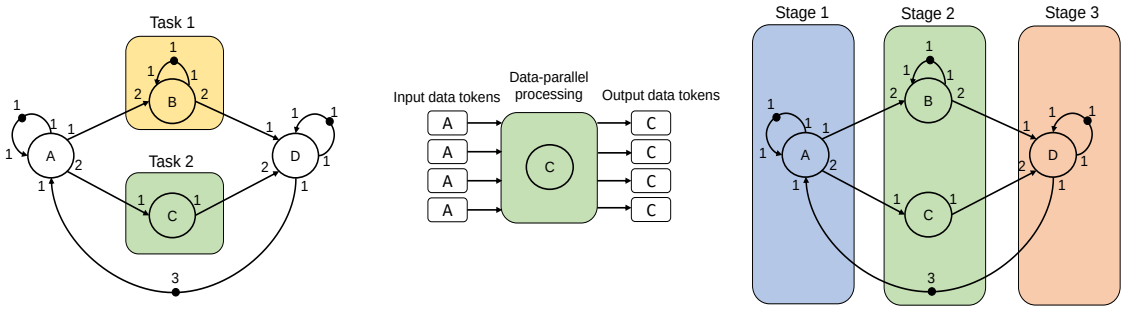
SDF graphs provide various mechanisms to model and exploit different levels of parallelism such as data, task and pipeline parallelisms in loop-intensive programs. Actually, the actors of a SDF graph that describes a loop-intensive program can be specified either as *stateful* or as *stateless*. A stateful actor is an actor whose execution instances are scheduled in a sequential order while a stateless actor is an actor whose execution instances are scheduled out of order, or in parallel across different PUs. These types of actors respectively enable to specify pipeline and data parallelisms in loop-intensive programs. A stateful actor is often described as a node with a self-loop channel, where the channel consists of a fixed number of tokens that represents the distance separating the successive execution instances of the stateful actor. Figure 2.6 depicts a loop-intensive program and its equivalent SDF graph.

The loop-intensive program of four instructions and four computing functions ($A$, $B$, $C$ and $D$). Each instruction is a logical and/or arithmetic operation that describes one or several dependency relations between the different invocations of the functions. For instance, the instruction $A[k] \leftarrow A[k-1] \odot D[k-3]$ is an arithmetic (or logical) operation that describes two dependency relations: a dependency from the $(k-1)^{th}$ invocation of actor A to the $k^{th}$ invocation of this actor and a dependency from the $(k-3)^{th}$ invocation of actor $D$ to the $k^{th}$ invocation of actor A. This loop-intensive program is graphically described by the SDF graph shown in Fig. 2.6b. This graph consists of four actors $(A, B, C, D)$, each describing a computing funtion of the program. Actors $A$, $B$ and $D$ are stateful actors while actor $C$ is a stateless actor. All of these actors are connected by a set of channels, each describing the flows of data exchanged between the computations, and some containing an initial number of tokens describing the distance separating the successive execution instances of connected actors. When deploying this graph on a multiprocessor architecture, data parallelism can be exploited by scheduling and mapping the execution instances of the actor $C$ on a data-parallel processor or on different PUs. At the same time, pipeline parallelism can be exploited by scheduling and mapping the execution instances of a stateful actor on different PUs in such a way that the successive executions of these execution instances can overlap over time. Alongside with data and pipeline parallelisms, task parallelism can also be exploited by scheduling and mapping the execution instances of actors $B$ and $C$ on different PUs. Figure 2.7 illustrates the exploitation of these three types of parallelisms. The joint exploitation of task, data and pipeline parallelisms is known to improve the performance achievable by loop-intensive applications [40]. Therefore, to ensure performance guarantee for CPS applications, it is important that the scheduling and mapping strategies for these applications exploit efficiently these three kinds of parallelisms.

The two most prominent performance measurements for applications modeled by SDF graphs are latency, which reflects the delay induced by a channel to transfer a token of data between dependent execution instances, and throughput, which indicates for each actor, the execution rate per unit time. From a CPS perspective, the study of these properties are of a high interest to characterize the timing behaviour of loop-intensive programs and to provide performance guarantee for CPS applications. Consequently, in this thesis, we focus on the characterization and analysis of these two performance metrics.

### 2.4.2 Scheduling under resource and communication constraints

An important design requirement for CPS applications is that, the heterogeneous multiprocessor architectures on which the computations are intended to be deployed, must contain




(a) Task Parallelism (b) Data Parallelism (c) Pipeline Parallelism

Figure 2.7: Types of parallelism exploitable in the SDF graph shown in figure 2.6b.

a finite number of processing and communication resources, and these resources have to be shared between the computations of one or more applications. This requirement can often lead to resource conflicts and/or communication bottlenecks when different computations need to access the same resources at the same time, and thus can cause a loss of parallelism and an deterioration of performance achievable by CPS applications. In order to prevent such a situation, there is a need of developing static scheduling and mapping strategies that can accommodate the resource and communication constraints of multiprocessor architectures to reduce in the early design stage of the applications, the over-allocation of resources needed for computations to meet their timing constraints. Since most of CPS applications are essentially loop-intensive applications, it is important that the scheduling strategies respect the loop-carried dependencies constraints of these application, and ensure that the performance achievable by a computation is not influenced by the other computations assigned to the same resources.

Scheduling computations with dependency relations on multiprocessor architectures under resource and/or communication constraints is a NP-complete problem well-known in the literature [29, 59]. Many authors have proposed several static scheduling approaches to tackle this problem with the goal of optimizing different performance metrics. A detailed survey of these approaches can be found in [29]. Among the existing works, there are a number [26, 24, 30, 38, 33, 48] that uses SDF graphs to tackle the problem. However, most of these works are limited to the scheduling of SDF graphs on homogeneous multiprocessor architectures. Among the works [25, 32, 58] tackling the scheduling and mapping of static dataflow graphs on heterogeneous multiprocessor architectures, an important number is limited to the scheduling of acyclic SDF graphs. Although these types of graphs can model many types of applications, they fail to model applications with cyclic dependencies. Since most of loop-intensive programs for CPS may contain cyclic depen-

dencies, the SDF graphs that describe these programs may consist of cycles like the SDF graph depicted in figure 2.6b. Consequently, a scheduling strategy for these graphs must deal with cyclic dependency constraints while satisfying both resource and communication constraints. Scheduling a cyclic SDF graphs on heterogeneous multiprocessor architectures under resource and communication constraints is a difficult problem rarely addressed in the literature. *The main goal of this thesis is to propose efficient static scheduling strategies to tackle this difficult problem while ensuring performance guarantees in terms of throughput and latency.*

## 2.5 Conclusion

In this chapter, we have presented the background and motivations that pushed us to be interested in the scheduling of SDF graphs on heterogeneous multiprocessor architectures under resource and communication constraints. In the next chapter we will review the basics of the SDF model and will present a succinct formulation of the main problem tackled by this thesis.

# CHAPTER 3

# State-of-the-Art and Problem Formulation

## Contents

## 3.1 Introduction

In this chapter we review the basics of the synchronous dataflow (SDF) model, which was introduced informally in the previous chapter, then we formulate the problem tackled by this thesis and we succinctly describe our main contributions. The chapter is organized as follows. Section 3.2 presents basic definitions and structural properties of SDF graphs. Section 3.3 reviews the static scheduling strategies for SDF models. Section 3.4 presents a succinct description of the main problem tackled by this thesis as well as the related works.

## 3.2 Synchronous Dataflow Graphs

### 3.2.1 Definition

A SDF graph is a multi-rate dependency graph $G_{sdf} = (V, E, P, C, M_0)$ where:

- $V$ is a finite set of nodes called actors.
- $E \subseteq V^2$ is a finite set of arcs representing First-in First-out (FIFO) channels.
- $P = \{p_e \in \mathbb{N}^* | \ e = (i, j) \in E\}$ is the set of production rates given by the function $p : E \longrightarrow \mathbb{N}^*$ that associates a production rate $p_e$ with each channel $e = (i, j) \in E$.
- $C = \{c_e \in \mathbb{N}^* | \ e = (i, j) \in E\}$ is the set of consumption rates given by the function $c : E \longrightarrow \mathbb{N}^*$ that associates a consumption rate $c_e$ with each channel $e = (i, j) \in E$.
- $M_0 = \{m_0(e) \in \mathbb{N} | \ e = (i, j) \in E\}$ is the set of initial markings given by the function $m_0 : E \longrightarrow \mathbb{N}$ that associates a fixed number $m_0(e)$ of tokens with each channel $e \in E$.

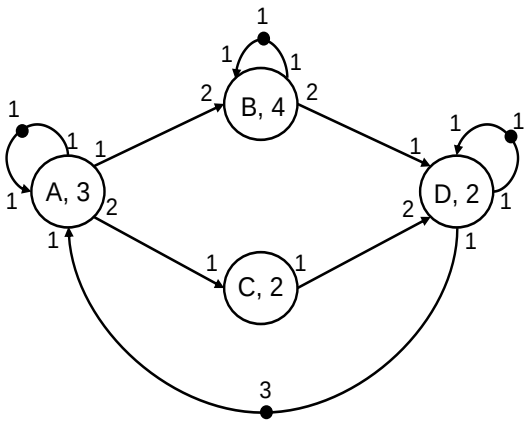



Figure 3.1: An example of timed SDF graph

**Timed Synchronous Dataflow Graph.** A SDF graph $G_{sdf}$ is said timed if there exists a function $\delta : V \rightarrow \mathbb{N}^*$ that associates a duration $\delta_i$ with every actor $i \in V$, where $\delta_i$ is the worst-case time to process a single execution instance of actor $i$. Figure 3.1 shows a graphical representation of a timed SDF graph, where each node models an actor and the worst-case processing time associated with the execution of each instance of this actor. For the rest of the manuscript, let us denote the execution instances of actors in terms of firings and let us denote the timed SDF graphs as $G^t_{sdf} = (V, E, P, C, M_0, \delta)$.

### 3.2.2 Consistency Analysis

Consistency is a property that has been introduced initially by Lee and Messerschmitt [61] to ensure that actors of a SDF graph can be statically scheduled with a bounded number of tokens. Let us consider a SDF graph $G_{sdf} = (V, E, P, C, M_0)$ and let $\Theta$ be the topology matrix of this graph, where $\Theta$ is a matrix of size $|E| \times |V|$ defined by:

$$\Theta_{ei} = \begin{cases} p_e & \text{if } e = (i, j), j \in V \\ -c_e & \text{if } e = (j, i), j \in V \\ 0 & otherwise \end{cases}$$

$G_{sdf}$ is said consistent if the rank of the matrix $\Theta$ is equal to $|V| - 1$. To illustrate this property, let us consider the SDF graph shown in figure 2.6b. The topology matrix of this graph is given by:

$$\Theta = \begin{array}{c} \begin{matrix} \text{A} & \text{B} & \text{C} & \text{D} \end{matrix} \\ \begin{bmatrix} 1 & -2 & 0 & 0 \\ 2 & 0 & -1 & 0 \\ 0 & 2 & 0 & -1 \\ 0 & 0 & 1 & -2 \\ -1 & 0 & 0 & 1 \end{bmatrix} \end{array} \begin{matrix} \\ \text{e=(A, B)} \\ \text{e=(A, C)} \\ \text{e=(B, D)} \\ \text{e=(C, D)} \\ \text{e=(D, A)} \end{matrix}$$

Considering this topology matrix, it is possible to check that $rank(\Theta) = |V| - 1 = 3$. This means that the SDF model described by this matrix is consistent. Actually, the consistency property ensures the existence of a minimum vector $q \in \mathbb{N}^{*|V|}$ with coprime components such that $\Theta.q^T = 0$. This vector is called the repetition vector and its components are called granularities or repetition factors. The granularity $q_i$ of an actor $i \in V$ corresponds to the minimum number of firings required for this actor to achieve a single iteration[1]. Thus, checking the consistency of a SDF model is equivalent to checking the existence of

[1] An *iteration* of a SDF graph is an execution sequence that brings back the graph to its initial state.

a repetition vector. For any SDF graph, the existence condition of a repetition vector is defined by:

$$p_e \times q_i = c_e \times q_j \qquad \forall e = (i, j) \in E. \tag{3.1}$$

Let us going back to the SDF graph depicted in figure 2.6b, if we want to check the existence of a repetition vector for this graph, we must try to solve the following system of equations:

$$\begin{cases} 1 \times q_A = 1 \times q_A \\ 1 \times q_A = 2 \times q_B \\ 2 \times q_A = 1 \times q_C \\ 1 \times q_B = 1 \times q_B \\ 2 \times q_B = 1 \times q_D \\ 1 \times q_C = 2 \times q_D \\ 1 \times q_D = 1 \times q_A \\ 1 \times q_D = 1 \times q_D \end{cases}$$

The solution of this system of equations gives a repetition vector $q = [q_A, q_B, q_C, q_D] = [2, 1, 4, 2]$ which proves that the SDF graph is consistent.

### 3.2.3 Liveness Analysis

Liveness is a property which ensures that every actor of a static dataflow model can be fired infinitely often without encountering deadlocks during the execution of the model. Liveness checking is a well-known problem extensively studied in the dataflow community. In 1971, Commoner et al. [64] have proposed the following theorem that serves as necessary and sufficient condition to ensure the liveness of marked graphs, which are named homogeneous synchronous dataflow (HSDF) graphs in the dataflow community.

**Theorem 3.1** (Commoner et al. [64])**.** *A HSDF graph is live if and only if the token count of every directed circuit is positive.*

Based on this theorem, Commoner et al. have proposed a polynomial-time algorithm —using depth-first search —to check the liveness of HSDF graph. The algorithm proceeds in two steps. In the first step, it removes every channel with non-zero marking from the HSDF graph and in the second step, it checks if resulting HSDF graph contains cycles. If the resulting graph is acyclic then the initial HSDF graph is live otherwise, it is not live.

**Algorithm 1:** Transform a consistent SDF graph into a HSDF graph

**Input:** a consistent SDF graph $G_{sdf} = (V, E, P, C, M_0)$ with a repetition vector $q$.

**Output:** an equivalent HSDF graph for $G_{sdf}$

Let $G_{hsdf} = (V', E', M_0')$ be an empty HSDF graph ;
**foreach** *actor* $i \in V$ **do**
  **for** $k = 1 \dots q_i$ **do**
    Add node $a_k$ to $V'$;
  **end**
**end**
**foreach** *channel* $e = (i, j) \in E$ **do**
  **for** $k' = 1 \dots q_j$ **do**
    $\pi_{ij}(k') \leftarrow \left\lceil \frac{k' \cdot c_e - m_0(e)}{p_e} \right\rceil$;
    $k \leftarrow (\pi_{ij}(k') - 1) \bmod q_i + 1$;
    Add arc $e' = (i_k, j_{k'})$ to $E'$ and set $m_0(e')$ to $-\left\lfloor \frac{\pi_{ij}(k') - 1}{q_i} \right\rfloor$ ;
  **end**
**end**
return $G_{hsdf}$;

Contrary to HSDF graphs, liveness checking for SDF graphs is a problem whose time complexity is not polynomial. Two techniques exist to check liveness for a consistent SDF graph. The first technique consists in transforming the SDF graph into an equivalent HSDF graph and applying the algorithm of Commoner et al. [64] to check liveness for the HSDF graph. Many polynomial-time algorithms exist in literature to transform a SDF graph into an equivalent HSDF graph [28, 34, 58]. Figure 3.2 shows an equivalent HSDF graph for the SDF graph depicted in figure 2.6b. This HSDF representation also called linear constraint graph, is obtained with the algorithm of de Groote et al. [28], which enables to generate for any consistent SDF graph, an equivalent HSDF graph with fewer channels. The different steps of this algorithm are explicitly detailed by Algorithm 1. Considering the equivalent HSDF graph, the algorithm of Commoner et al. [64] can easily be applied to check liveness. For this liveness checking technique, it is important to mention that the transformation of a SDF graph to an equivalent HSDF graph may lead sometimes to a graph of exponential size, and thus can make exponential the time and space complexity of this technique. The second technique for liveness checking [39] consists in performing the symbolic execution of the SDF graph —i.e. to execute all the actors of the model exactly as many times as indicated by the repetition vector —until the graph reaches a repetitive

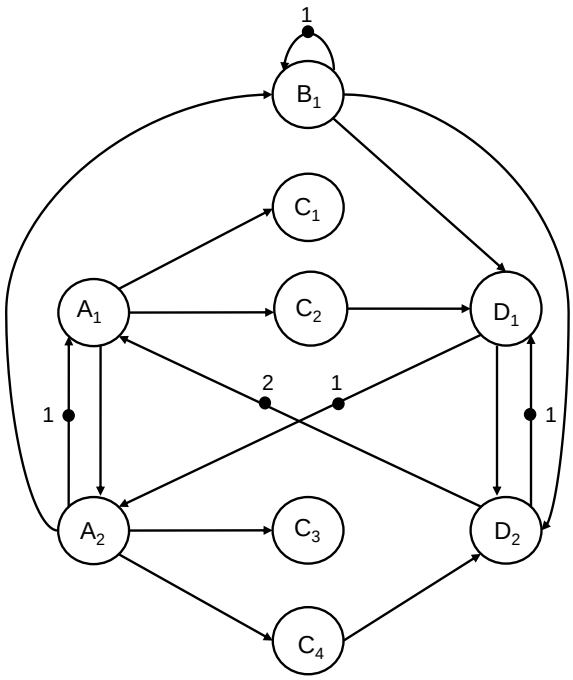



Figure 3.2: Equivalent HSDF graph for the SDF graph shown in figure 2.6b. The notation $x_y$ in the nodes denotes the $y^{th}$ firing of a actor $x$, where $y \in [1, q_x]$, $q_x$ being the granularity of the actor $x$.

execution pattern. Whether the symbolic execution of a SDF graph enables to perform a least one iteration without encountering deadlocks, then the SDF graph is said to be live. Figure 3.3 shows the symbolic execution trace of the SDF graph depicted in figure 2.6b. As it can be noted, the SDF graph can iterate several times without encountering deadlocks. This means that the graph is live. Symbolic execution of a SDF graph has a low space complexity compared to the first technique, however, checking the liveness of a SDF graph with symbolic execution could imply an exponential number of computations which makes the time-complexity of this technique also exponential.

In order to get around the complexity issue of liveness checking techniques presented above, Marchetti and Munier [35] have established the following theorem, which serves as sufficient condition to identify live SDF graphs.

**Theorem 3.2** (Marchetti and Munier [35]). *Let $G_{sdf} = (V, E, P, C, M_0)$ be a normalized SDF graph. $G_{sdf}$ is said live if for every directed circuit $\mu$, the following condition holds:*

$$\sum_{\forall e=(i,j)\in\mu} m_0(e) > \sum_{\forall e=(i,j)\in\mu} c_e - gcd(p_e, c_e)$$

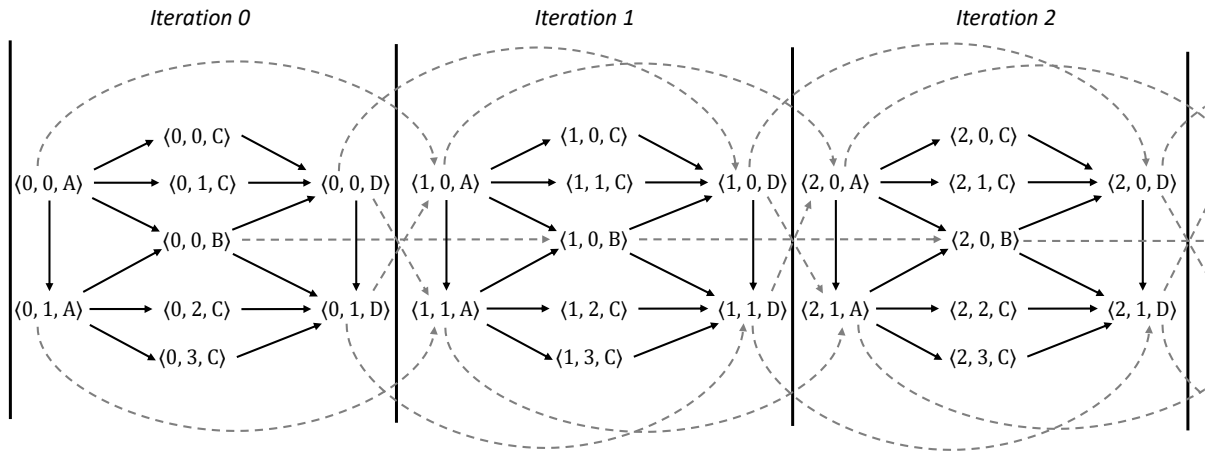



Figure 3.3: Symbolic execution trace of the SDF graph of Fig. 2.6b. Solid arcs are intra-iteration dependencies, dashed arcs are inter-iteration dependencies and the notation $\langle n, k_i, i\rangle$ stands for the completion of the $k_i^{th}$ firing of an actor $i$ in the $n^{th}$ iteration of the SDF graph, where $n \in \mathbb{N}$, $k_i \in [0, q_i)$, $q_i$ being the granularity of the actor $i$.

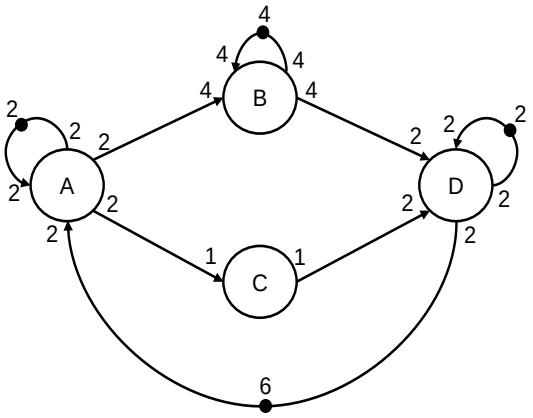



Figure 3.4: Normalized representation of the SDF graph depicted in figure 2.6b

where $gcd(p_e, c_e)$ is the greatest common divisor of $p_e$ and $c_e$. As it can be noted, Theorem 3.2 is based on the normalized SDF graphs. A SDF graph is said normalized if for every actor the consumption and production rates are identical. Normalization is a concept introduced by Marchetti and al. [35] to simplify the liveness analysis of SDF graphs. Let $G_{sdf} = (V, E, P, C, M_0)$ be a consistent SDF graph with a repetition vector $q \in \mathbb{N}^{*(|V|)}$ and let $lcm_q$ be the least common multiple of the components of $q$. For every channel $e = (i, j) \in E$, let us denote by $n_e = \frac{lcm_q}{q_i \cdot p_e}$ (or $n_e = \frac{lcm_q}{q_j \cdot c_e}$) the normalization factor of the channel $e$ where $lcm_q$ is the least common multiple of the vector $q$. The normalized representation of $G_{sdf}$ is obtained by multiplying the weights (i.e. production rate, marking and consumption) of every channel by its corresponding normalization factor. Figure 3.4 shows the normalized representation of the SDF graph shown in figure 2.6b.

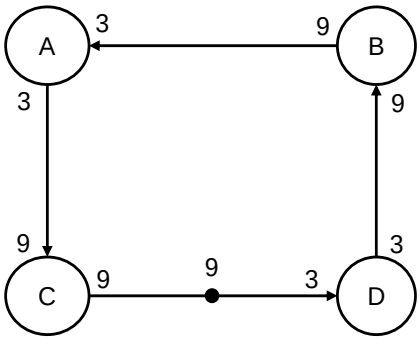



Figure 3.5: Counter example showing that theorem 3.2 is not a necessary condition for liveness.

Using this normalized SDF graph, Theorem 3.2 can easily be applied with a polynomial-time algorithm [35] based on depth first search to ensure the existence of a live marking. It is important to remind that Theorem 3.2 is actually a sufficient condition that ensures the liveness property for SDF graphs but it is not a necessary condition. As counter example, figure 3.5 shows a normalized SDF graph for which Theorem 3.2 does not hold since $\sum_{\forall e=(i,j)\in\mu} m_0(e) = 9$ , and $\sum_{\forall e=(i,j)\in\mu} c_e - gcd(p_e, c_e) = 12$; However, a live marking can be obtained by the execution sequence DDDBAAAC that achieves an iteration of the graph.

## 3.3 Static Scheduling of Synchronous Dataflow Graphs

Static scheduling of SDF graphs have been the object of many studies in the dataflow community. Scheduling a SDF graph consists in associating a starting time to the firings of each actor according to a given strategy that does not violate the precedence constraints imposed by the channels. A SDF graph can be statically scheduled if and only if it is consistent and live. Consistency and liveness are the fundamental properties ensuring that a SDF graph can be executed repetitively with a bounded number of tokens without encountering deadlocks. This section reviews the basics of static scheduling techniques for SDF graphs.

### 3.3.1 Basic Definitions and Theorems

**Definition 3.1** (Schedule). *Let $G^t_{sdf} = (V, E, P, C, M_0, \delta)$ be a consistent and live SDF graph. A schedule of $G^t_{sdf}$ is a function $\sigma : \mathbb{N} \times V \rightarrow \mathbb{Q}^+$ that associates a starting time $\sigma(k_i, i)$ with every firing $\langle k_i, i \rangle$ of every actor $i \in V$.*

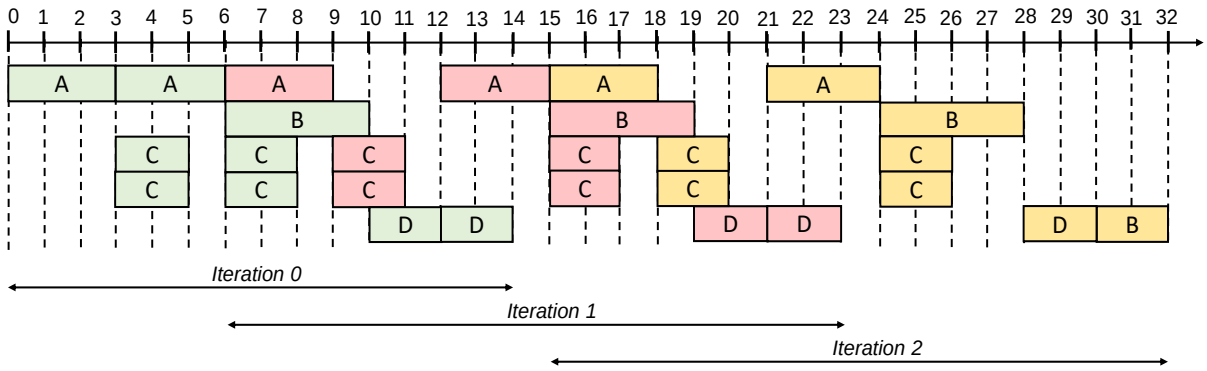



Figure 3.6: A self-timed schedule of the timed SDF graph depicted in figure 3.1.

**Definition 3.2** (Admissible Schedules)**.** *Let $G^t_{sdf} = (V, E, P, C, M_0, \delta)$ be a consistent and live SDF graph. A schedule $\sigma$ is said admissible for $G^t_{sdf}$ if and only if it fulfils the precedence constraints imposed by every channel of $G^t_{sdf}$. More precisely, for every channel $e = (i, j) \in E$ and every couple $(k_i, k_j) \in \mathbb{N}^2$, if the channel $e$ induces a dependency relation from a firing $\langle k_i, i\rangle$ to a firing $\langle k_j, j\rangle$ then, the following constraint must be fulfilled by any admissible schedule $\sigma$.*

$$\sigma(k_j, j) \geq \sigma(k_i, i) + \delta_i \tag{3.2}$$

### 3.3.2 Self-timed Schedules Versus Periodic Schedules

Admissible schedules for SDF graphs can be classified into self-timed and periodic schedules. In a self-timed schedule, the firing of an actor is executed as soon as the necessary tokens of data are available while in a periodic schedule, the firing of an actor is executed according to a specific time period often called initiation interval.

**Definition 3.3** (self-timed schedule)**.** *Let $G^t_{sdf} = (V, E, P, C, M_0, \delta)$ be a consistent and live SDF graph, let $\sigma$ be an admissible schedule for $G^t_{sdf}$ and let $\mathcal{P}(E) \subseteq \mathbb{N}^2$, be a set of tuples $(k_i, k_j)$ such that $\forall e = (i, j) \in E$, there exists a dependency relation from $\langle k_i, i\rangle$ to $\langle k_j, j\rangle$. The schedule $\sigma$ is said self-timed if and only if the following conditions holds.*

- *Condition 1:* $\sigma(k_i, i) \geq 0, \ \forall i \in V, \ \forall k_i \in \mathbb{N}$.
- *Condition 2:* $\sigma(k_j, j) \geq \sigma(k_i, i) + \delta_i, \ \forall e = (i, j) \in E, \ \forall (k_i, k_j) \in \mathcal{P}(E)$.

The self-timed schedule of any SDF graph consists of a finite sequence of firings, called the **transient phase** followed by a sequence of firings which is repeated infinitely often and is called the **periodic phase**. Figure 3.6 depicts a self-timed schedule of the timed

SDF graph presented in figure 3.1. This schedule shows three iterations of the graph, each described with a specific colour. We recall that the iteration of a SDF graph is the sequence of firings that brings back the graph to its initial state. The sequence of firings in the first iteration describes the transient phase while the sequences of firings in the other iterations describe the periodic phase. In this schedule, the number of processing resources that execute the firings of actors is assumed to be unbounded and the firings of stateless actors can be executed in parallel on different processing resources. Self-timed schedule is a scheduling strategy that achieves optimal throughput and provide performance guarantee for applications modeled by SDF graphs [34, 38]. However, the time complexity to compute the sequences of firings describing the transient phase is commonly admitted to be exponential and difficult to evaluate. This makes the computation of self-time schedules exponential and difficult to implement.

In order to surround the implementability complexity of self-timed schedules, r-periodic schedules [37] have been introduced.

**Definition 3.4** (Periodic Schedules)**.** *Let $G^t_{sdf} = (V, E, P, C, M_0, \delta)$ be a consistent and live timed SDF graph and let $\sigma$ be an admissible a schedule for $G^t_{sdf}$. The schedule $\sigma$ is said periodic with period $\lambda \in \mathbb{Q}^{+*}$, if for any actor $i \in V$ there exists $s_i \in \mathbb{Q}^+$ such that the following set of equations hold:*

$$\sigma(k_i, i) = s_i + k_i \cdot \lambda, \qquad \forall k_i \in \mathbb{N} \tag{3.3}$$

where $s_i$ is the time at which an actor $i$ must be scheduled to fire and $\lambda$ is the time period between two successive iterations of $G^t_{sdf}$ in the schedule $\sigma$. In the class of periodic schedules, software pipelined (SWP) schedules [49, 50, 51, 60] are of a hight interest.

**Definition 3.5** (SWP Schedules)**.** *Let $G^t_{sdf} = (V, E, P, C, M_0, \delta)$ be a consistent and live SDF graph and $\sigma$ be a schedule of $G^t_{sdf}$. The schedule $\sigma$ is said software pipelined with period $\lambda$ if the following set of constraints hold:*

$$\sigma(k_i + n \cdot q_i, i) = \sigma(k_i, i) + n \times \lambda, \qquad \forall i \in V,\ \forall k_i \in [0, q_i),\ \forall n \in \mathbb{N} \tag{3.4}$$

Actually, equation (3.4) defines a set of cyclicity constraints where $\sigma(k_i + n \cdot q_i, i)$ is the

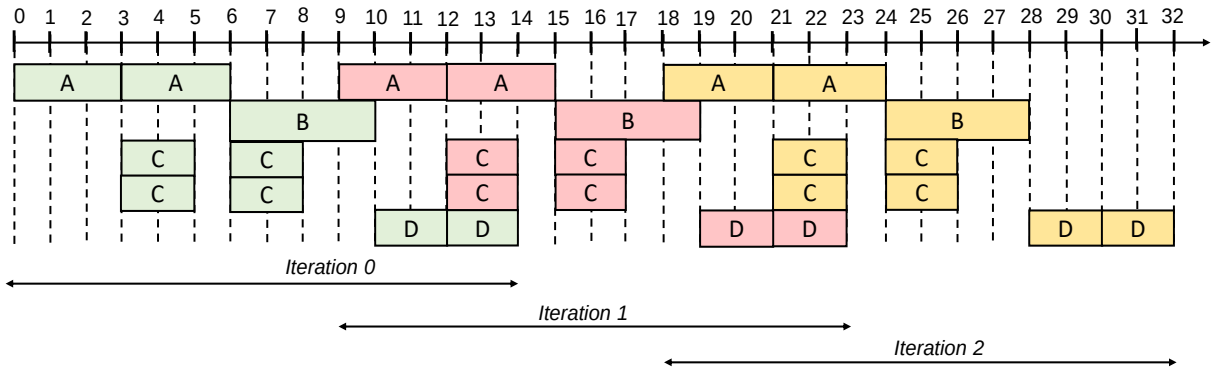



Figure 3.7: A SWP Schedule for the timed SDF graph depicted in figure 3.1.

time at which the firing $\langle k_i, i\rangle$ is scheduled in the $n^{th}$ iteration of $G_{sdf}$ and $\sigma(k_i, i) \in \mathbb{Q}^+$ the time at which the firing $\langle k_i, i\rangle$ must be scheduled to start. Figure 3.7 depicts a SWP schedule of the timed SDF graph shown in figure 3.1. In this schedule, the number of processing resources that execute the firings of actors is assumed to be unlimited and the firings of stateless actors can be executed in parallel on different processing resources. As it can be noted, the firings of each actor are executed according to a time period ($\lambda = 9$) that enables to overlap the successive iterations of the SDF graph. Like self-timed schedules, SWP schedules achieve maximum throughput for applications modeled by SDF graphs; however, they are easier to implement than self-time schedules. In this thesis, we will focus on the SWP scheduling strategy to deploy the loop-intensive programs modeled by SDF graphs on heterogeneous multiprocessor architectures.

### 3.3.3 Throughput Evaluation

Let $G^t_{sdf} = (V, E, P, C, M_0, \delta)$ be a live and consistent SDF graph and let $\sigma$ be an SWP schedule of period $\lambda$ for $G^t_{sdf}$. The throughput of an actor $i \in V$ is the average number of firings occurred within a time interval in the schedule $\sigma$ and the throughput of $G^t_{sdf}$ is defined as the average number of stable iterations initiated per time unit in the schedule $\sigma$. More formally, if we denote by $\beta_i$ the throughput of actor $i \in V$, then $\beta_i$ is given by:

$$\beta_i = \lim_{k_i \to \infty} \frac{k_i}{\sigma(k_i, i)} \tag{3.5}$$

Now, if we denote by $\beta$ the throughput achievable by $G_{sdf}$ in the schedule $\sigma$ then $\beta = \min_{\forall i \in V}\{\beta_i\}$. Since, the period $\lambda$ is the average time elapsed between two successive iterations of $G^t_{sdf}$, the throughput of $G^t_{sdf}$ can be defined as the inverse of the period $\lambda$. For instance, in the SWP schedule shown in figure 3.7, one can note that new iteration of $G^t_{sdf}$ occurs

every 9 time units. Thus $\lambda = 9$ and $\beta = 1/9$. As $\lambda$ is proportional to $\beta$, we will note that the schedules that minimize the period $\lambda$ for a SDF graph maximize implicitly the throughput $\beta$.

It is well known that the period $\lambda$ achievable by a SDF graph in a given schedule, is governed by the loop-carried dependencies induced by the channels of this graph. In order to handle these dependencies, the SDF graph is often transformed into an equivalent HSDF graph which exhibits all the dependency relations between the different firings of actors. Using the equivalent HSDF graph, the period $\lambda$ is given by:

$$\lambda = \max_{\forall C \in cycles} \frac{d(C)}{m(C)} \tag{3.6}$$

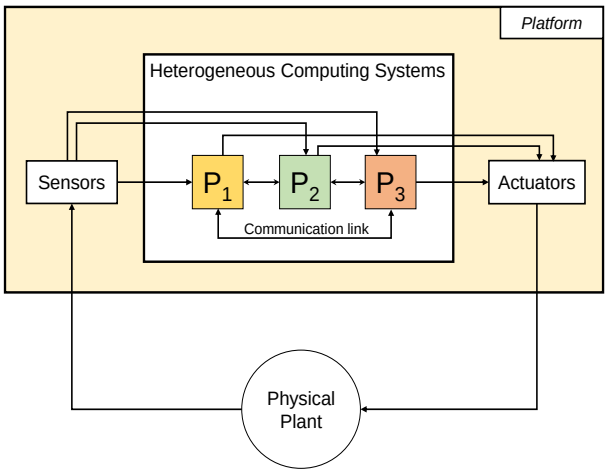
where $d(C)$ is the sum of delays (i.e the sum of WCETs) of nodes in a cycle $C$ of the HSDF graph, and $m(C)$ is the sum of tokens around $C$ [34, 44, 49]. The minimum period achievable by the HSDF graph is then given by the cycles with the minimum value of $\lambda$.

### 3.3.4 Latency Evaluation

The latency $\mathcal{T}$ of a SDF graph is the maximum time required to complete a single iteration of this graph. In other words, this time corresponds to the response time of the SDF graph in a given schedule. Going back to the timed SDF graph of our running example (refer to figure 3.1) and the SWP schedule of this SDF graph (refer to figure 3.7), one can note that the length of the maximum time interval required to achieve a single iteration of the graph is $\mathcal{T} = 14$, which corresponds to the latency (or the response time) of this graph. The minimum latency achievable by a SDF graph depends on the minimum latency induced by the channels of this graph. The latency induced by a channel is generally defined in terms of firings. Let $G^t_{sdf} = (V, E, P, C, M_0, \delta)$ be a consistent and live SDF graph and let $\sigma$ be a SWP schedule of $G_{sdf}$. Let $(k_i, k_j)$ be a couple of positive integers. If the channel $e$ induces a dependency relation from $\langle k_i, i\rangle$ to $\langle k_j, j\rangle$ then, the latency induced by the channel between these firings is the time elapsed between the end of $\langle k_i, i\rangle$ and the beginning of $\langle k_j, j\rangle$. More formally, this latency is given by:

$$\mathcal{T}_{k_i k_j} = \sigma(k_j, j) - \sigma(k_i, i) - \delta_i \tag{3.7}$$

Using equation 3.7, one can characterize the set of admissible schedules that achieve minimum latency for $G^t_{sdf}$. We will note that the schedules that minimize latency for a SDF graph do not necessarily maximize throughput for this graph. In the next chapter, we will




(a) A CPS structure

Computation costs matrix

| | A | B | C | D |
|---|---|---|---|---|
| $P_1$ | 1 | 3 | 2 | 3 |
| $P_2$ | 5 | 2 | 1 | 2 |
| $P_3$ | 3 | 1 | 6 | 1 |

Communication costs matrix

| | $P_1$ | $P_2$ | $P_3$ |
|---|---|---|---|
| $P_1$ | 0 | 1 | 1 |
| $P_2$ | 1 | 0 | 2 |
| $P_3$ | 1 | 2 | 0 |

(b) Cost Matrices for executing the SDF graph shown in figure 2.6b on the heterogeneous computing system

Figure 3.8: An example of architecture

show how to characterize the SWP schedules that achieve minimum latency for a timed SDF graph as well as the SWP schedules that achieve maximum throughput.

## 3.4 Problem Formulation and Related Works

In this section, we formulate and illustrate the main problem tackled by this thesis and we present related works.

Let $G_{sdf} = (V, E, P, C, M_0)$ be a cyclic, consistent, live and non-timed SDF graph, that describes a loop-intensive application and let $G_{hma} = (R, \Delta, \Gamma)$ be an heterogeneous multi-processor architecture (HMA) on which $G_{sdf}$ is intended to be deployed, where:

- $R$ is a finite set of heterogeneous processing units (PUs), each connecting with the other ones through logic communication links.
- $\Delta$ is a matrix of size $|R| \times |V|$ that specifies the computation costs $\Delta_{xi}$, where $\Delta_{xi}$ is the worst-case execution time of a single firing of an actor $i \in V$ on a PU $x \in R$.
- $\Gamma$ is a matrix of size $|R| \times |R|$ that specifies the communication costs $\Gamma_{xy}$, where $\Gamma_{xy} = \Gamma_{yx}$ is the worst-case delay to transmit a single token of data from $x$ to $y$. Note that if $x = y$ then $\Gamma_{xy} = \Gamma_{yx} = 0$ otherwise $\Gamma_{xy} \neq 0$ and thus $\Gamma_{yx} \neq 0$.

**Problem.** Assuming the SDF graph $G_{sdf}$ and the architecture model $G_{hma}$, the problem tackled here is to find an optimal SWP schedule $\sigma$ of period $\lambda$ for $G_{sdf}$ —i.e. a SWP schedule that achieves maximum throughput —under the following constraints:

- *cyclicity constraints*: each firing of each actor $i \in V$ is executed cyclically on the PUs of $G_{hma}$ according to a period $\lambda$ such that:

$$\sigma(k_i + n \cdot q_i, i) = \sigma(k_i, i) + n \times \lambda, \qquad \forall i \in V,\ \forall k_i \in [0, q_i),\ \forall n \in \mathbb{N}$$

  where $q_i$ is the granularity of actor $i \in V$.

- *resource constraints*: each firing of each actor $i \in V$ can only be assigned to a single PU $x \in R$ and the execution of two firings assigned to this PU cannot overlap.

- *precedence and communication constraints*: for any $(k_i, k_j) \in \mathbb{N}^2$ and any channel $e = (i, j) \in V$, if the channel $e$ induces a dependency relation from a firing $\langle k_i, i\rangle$ to a firing $\langle k_j, j\rangle$ and if these firings are respectively assigned to the PUs $x, y \in R$, such that $x \neq y$, then, there exists a non-zero communication costs $\Gamma_{xy}$ between these firings and the following inequality is satisfied:

$$\sigma(k_j, j) \geq \sigma(k_i, i) + \Delta_{xi} + \Gamma_{xy}.$$

In order to illustrate this problem, let us consider the system architecture depicted in figure 3.8. This architecture describes a cyber-physical system (CPS), which is a feedback control system. This system consists of sensors, actuators and a heterogeneous computing system. The heterogeneous computing system is a multiprocessor computing system composed of three heterogeneous PUs, each connected with the other ones through communication links that enable data parallel transmission, and each offering a specific performance to execute the computations of the CPS applications. Each pair of PUs is connected with a single communication link, which has a specific cost for data transmission. Let us assume that this computing system is intended to execute the loop-intensive program described by the non-timed SDF graph of our running example (refer to figure 2.6b). The computation and communication costs matrices for executing the actors of these graphs are given by figure 3.8b. Considering these matrices, we are looking for an optimal SWP that achieves maximum throughput for SDF graph under the resource and communication constraints of the architecture considered.

Readers who are familiar with the literature of software pipelining of loop-intensive programs [71, 70, 68, 33, 47, 48, 49, 52, 55, 57, 60] will find the optimality objective of this problem very ambitious because of the consideration of resource constraints, which make NP-hard, the complexity to solve this problem. Whether communication constraints are considered further, the space of solutions for the problem is reduced and thus, the

Table 3.1: Related works on SWP scheduling of loops modeled by dataflow graphs

| Litterature | Multi-rate graphs | Cyclic dependencies | Hetero. architectures | Resource constraints | Com. constraints |
|---|---|---|---|---|---|
| Feautrier et al. [47] | ✗ | ✓ | ✓ | ✓ | ✗ |
| Govindarajan et al. [49] | ✗ | ✓ | ✓ | ✓ | ✗ |
| Hanen and Munier, [52] | ✗ | ✗ | ✗ | ✓ | ✗ |
| Wang et Eisenbeis, [53] | ✗ | ✓ | ✗ | ✓ | ✗ |
| Robert et al. [45] | ✗ | ✓ | ✗ | ✓ | ✗ |
| Gasperoni et al., [48] | ✗ | ✓ | ✗ | ✓ | ✗ |
| Udupa et al, [33] | ✓ | ✓ | ✗ | ✓ | ✗ |
| Hatanaka et al., [70] | ✓ | ✗ | ✗ | ✓ | ✗ |
| [Wei et al., 2012][68] | ✓ | ✗ | ✗ | ✓ | ✓ |
| Jiang et al., [71] | ✓ | ✗ | ✓ | ✓ | ✗ |
| This thesis | ✓ | ✓ | ✓ | ✓ | ✓ |

complexity to find an optimal solution is NP-hard in the strong sense. Consequently, to solve efficiently the problem, heuristics are necessary. However, before designing efficient heuristics, we need to characterize the set of optimal solutions. For this purpose, the problem can be formulated as an integer linear program (ILP), which can be solved by means of ILP solvers to generate optimal solutions even in non-polynomial time. Based on the performance achievable by the ILP solver, we could thus, characterize efficient scheduling heuristics for the problem.

### 3.4.1 ILP-based Scheduling Approaches

During the past decades, a variety of ILP formulations have been proposed to tackle the resource-constrained SWP scheduling problems of loop-intensive programs.

Feautrier et al. [47] and Govindarajan et al. [49] have proposed discrete-time ILP formulations for solving the resource-constrained SWP scheduling of loop-intensive programs. These ILP formulations hold both for architectures with homogeneous resources and those with heterogeneous resources. However, none of the proposed ILP models consider communication constraints, which are inherent to the SWP scheduling problem tackled by this thesis. Moreover, in their ILP formulations, Feautrier [47] and Govindarajan et al. [49] assumed that the loop-intensive programs are described with single-rate dependency graphs —i.e HSDF graphs —, which are particular cases of SDF graphs (refer to the previous chapter). Although SDF graphs can be converted into HSDF graphs, it could be interesting to avoid the conversion costs of SDF graphs by formulating an ILP model that operates directly on the SDF graphs.

Recently, Udupa et al. [33] have proposed an ILP formulation to tackle the resource-constrained SWP scheduling problem of SDF graphs on multiprocessor architectures with

graphical processing units (GPUs). Although this ILP formulation operates directly on SDF graphs and hold both the architectures with homogeneous or heterogeneous GPUs, it does not consider communication constraints. In this thesis, we aim at proposing a new ILP that operates directly on SDF graphs to optimize throughput while considering both resource, cyclicity and communication constraints. To the best of our knowledge, there is no work in the current literature that proposes such an ILP formulation.

### 3.4.2 Scheduling Heuristics

Aside the ILP-based scheduling techniques, we have investigated the existing heuristics for tackling the resource-constrained SWP scheduling of static dataflow graphs. Indeed, there exists a number of heuristics in the literature, that deal with the resource-constrained SWP scheduling of loop-intensive programs modeled by single-rate dependency graphs.

Hanen and Munier [52] have proposed heuristics for acyclic dependency graphs, to derive resource-constrained SWP schedules on architectures with homogeneous resources. Basically, the idea of these heuristics consists of two steps. The first step is to add some arcs to the acyclic dependency graphs. These new arcs link nodes to be executed on the same group of processing resources. The second step is to partition the processing resources into groups and to pipeline nodes on these resources. The proposed heuristics look quite powerful, although not guaranteed.

Wang and Eisenbeis [53], Robert et al. [45], Gasperoni and Schwielgelshohn [48] have introduced decomposed SWP scheduling heuristics, to derive resource-constrained SWP schedules for cyclic dependency graphs. Given a loop modeled by a single-rate cyclic dependency graph, the general idea of decomposed SWP scheduling heuristics consists of four steps. In the first step, the cyclic dependency graph is scheduled under unlimited resources i.e. without resource constraints. In the second step, some information from the schedule obtained in the first step is available to delete some arcs in the cyclic dependency graph so as to obtain an acyclic dependency graph. This acyclic dependency graph is scheduled under resource constraints in the third step using a list scheduler. Finally, in the fourth step, a resource-constrained SWP schedule is deduced using the information obtained in the schedules performed in the first and third steps. Although these heuristics are guaranteed, none of them consider neither architectures with heterogeneous resources, nor with communication constraints.

### 3.4.3 This Work

Table 3.1 summarizes the related works previously presented and it highlights the criteria considered by these works to schedule loop-intensive applications modeled by static dataflow graphs. In this work we consider both cyclicity, resource, precedence and communication constraints to schedule loop-intensive applications modeled by SDF graphs on heterogeneous multiprocessor architectures. To the best of our knowledge, there is no work in the current literature that tackles such a scheduling problem. In order to solve this problem, we propose:

- An ILP model to characterize the set of optimal SWP schedules that maximizes throughput for SDF graphs on heterogeneous multiprocessor architectures. Our ILP model accommodates both cyclicity, resource, precedence, communication constraints and it exploits efficiently task, data and pipeline parallelisms specified in the application graphs.
- A SWP scheduling heuristic based upon the heuristic of Gasperoni and Schwiegelshohn, that generates approximated SWP scheduling solutions for the problem.

## 3.5 Conclusion

In this chapter, we have reviewed the basics of SDF models and the static scheduling strategies to execute these models. We have also presented a detailed description of the main problem tackled by this thesis, as well as the related works. In the next part of the manuscript, we will detail explicitly our contributions.

# Part II

# Contributions

# CHAPTER 4

# Software Pipelined Scheduling of Timed Synchronous Dataflow Models

## Contents

## 4.1 Introduction

In this chapter, we characterize the software pipelined (SWP) schedules that achieve optimal throughput/latency for timed SDF graphs. Characterizations made in this chapter will be extended in the next chapter to provide an integer linear programming formulation for the SWP scheduling problem of SDF graphs under resource and communication constraints. The current chapter is organized as follows. In section 4.2, we characterize the set of admissible SWP schedules for timed SDF graphs. In section 4.3, we characterize and compute the SWP schedules that achieve maximum throughput for timed SDF graphs. In section 4.4 we characterize the SWP schedules that achieve minimum latency for timed SDF graphs and we conclude in section 4.5.

## 4.2 Characterization of Admissible SWP Schedules

Let us consider a timed SDF graph $G^t_{sdf} = (V, E, P, C, M_0, \delta)$ with a repetition vector $q$ and let us assume that $G^t_{sdf}$ is live and consistent. According to definition 3.2, a schedule is said admissible for $G^t_{sdf}$ if and only if it fulfils the precedence constraints imposed by every channel of $G^t_{sdf}$. In order to characterize the set of precedence constraints that must be fulfilled by the set of admissible SWP schedules for $G^t_{sdf}$, we first need to define and formalize the dependency relations induced by every channel in $G^t_{sdf}$. For this purpose, let us consider the following notations. For each actor $i \in V$, $k_i$, $q_i$ and $\delta_i$ represent respectively the firing index, the granularity and the worst-case execution time to process a single firing of this actor. For any channel $e = (i, j) \in E$, $p_e$, $m_0(e)$ and $c_e$ are respectively the production rate, the initial marking and the consumption rate of the channel $e$.

### 4.2.1 Dependency relations induced by channels

**Definition 4.1** (Dependency Relation)**.** *Let us consider a channel $e = (i, j) \in E$ and let $(k_i, k_j) \in \mathbb{N}^2$. The channel $e$ induces a strict dependency relation from a firing $\langle k_i, i\rangle$ to a firing $\langle k_j, j\rangle$ if and only if each of the following conditions hold:*

- *Condition 1: $\langle k_j, j\rangle$ cannot be executed before the end of $\langle k_i, i\rangle$.*
- *Condition 2: $\langle k_j, j\rangle$ can be executed before the end of $\langle k_i + 1, i\rangle$.*
- *Condition 3: $\langle k_j - 1, j\rangle$ can be executed before the end of $\langle k_i, i\rangle$.*

In order to formalize definition 4.1, we have established the following lemma.

**Lemma 4.1.** *Let us consider a channel $e = (i, j) \in E$ and let $(k_i, k_j) \in \mathbb{N}^2$. The channel $e$ induces a strict dependency relation from a firing $\langle k_i, i\rangle$ to a firing $\langle k_j, j\rangle$ if and only if the following inequalities hold:*

$$c_e > m_0(e) + k_i \cdot p_e - k_j \cdot c_e \geq max\{0, c_e - p_e\}.$$

*Proof.* Let $e = (i, j) \in E$ be a channel and let $(k_i, k_j) \in \mathbb{N}^2$. After the execution of firings $\langle k_i, i\rangle$ and $\langle k_j, j\rangle$, the number of tokens remaining on the channel $e$ is equal to $m_0(e) + (k_i + 1) \cdot p_e - (k_j + 1) \cdot c_e$. Using this characterization, we can formalize the conditions stated in definition 4.1.

- *Condition 1* can be reformulated as follows: After the execution of the firing $\langle k_i - 1, i\rangle$, there are not enough tokens on the channel $e$ so that the firing $\langle k_j, j\rangle$ may be executed. By formalizing this assumption, we get the following inequality which ensures that if $\langle k_j, j\rangle$ is executed just after $\langle k_i - 1, i\rangle$ and before the end of $\langle k_i, i\rangle$, then, the number of tokens remaining on the channel $e$ is strictly lower to 0.

$$m_0(e) + k_i \cdot p_e - (k_j + 1) \cdot c_e < 0 \tag{4.1}$$

- *Condition 2* can be reformulated as follows: After the execution of $\langle k_j, j\rangle$ and $\langle k_i + 1, i\rangle$, the number of tokens remaining on the channel $e$ is greater or equal to 0. By formalizing this assumption, we get the following inequality:

$$m_0(e) + (k_i + 1) \cdot p_e - (k_j + 1) \cdot c_e \geq 0 \tag{4.2}$$

- *Condition 3* can be reformulated as follows: After the execution of $\langle k_j - 1, j\rangle$ and $\langle k_i, i\rangle$, the number of tokens remaining on the channel $e$ is greater of equal to 0. By formalizing this assumption we get the following inequality:

$$m_0(e) + k_i \cdot p_e - k_j \cdot c_e \geq 0 \tag{4.3}$$

Combining equations (4.1), (4.2) and (4.3), we get:

$$c_e > m_0(e) + k_i \cdot p_e - k_j \cdot c_e \geq max\{0, c_e - p_e\}.$$ ■

Actually, lemma 4.1 is a necessary and sufficient condition that ensures the existence of a dependency relation between any couple of firings. A similar lemma has been established

by Marchetti and Munier[36] for $k_i, k_j \in \mathbb{N}^*$. Let us consider for instance, the SDF graph shown in figure 2.6b and let us check if the channel $e = (C, D)$ induces a dependency relation from $\langle 1, C\rangle$ to $\langle 0, D\rangle$. Applying lemma 4.1, we get $2 > 0 + 1 - 0 \geq max\{2 - 1, 0\}$, which implies the existence of a dependency relation from $\langle 1, C\rangle$ to $\langle 0, D\rangle$.

Now, to identify the dependency relations induced by every channel of $G^t_{sdf}$ in a SWP schedule, we have established lemma 4.2. Before presenting this lemma, let us denote by $gcd_e$ the greatest common divisor of the production rate $p_e$ and the consumption rate $c_e$ of a channel $e = (i, j) \in E$ and let $X^{min}_{k_i k_j}$, $X^{max}_{k_i k_j} \in \mathbb{Z}$ be two variables given by:

$$X^{min}_{k_i k_j} = \left\lceil \frac{max\{0, c_e - p_e\} - m_0(e) - k_i \cdot p_e + k_j \cdot c_e}{q_i \cdot p_e} \right\rceil$$

$$X^{max}_{k_i k_j} = \left\lfloor \frac{c_e - gcd_e - m_0(e) - k_i \cdot p_e + k_j \cdot c_e}{q_i \cdot p_e} \right\rfloor$$

such that $X^{max}_{k_i k_j} \geq X^{min}_{k_i k_j}$, where $k_i$, $k_j \in \mathbb{N}$.

**Lemma 4.2.** *Let $e = (i, j) \in E$ be a channel of $G^t_{sdf}$ and let $\langle k_i + n \cdot q_i, i\rangle$ and $\langle k_j + n' \cdot q_j, j\rangle$ be two firings, where $k_i \in [0, q_i)$, $k_j \in [0, q_j)$ and $n, n' \in \mathbb{N}$.*

1. *If the channel $e$ induces a dependency relation from $\langle k_i + n \cdot q_i, i\rangle$ to $\langle k_j + n' \cdot q_j, j\rangle$ then, there exists $X \in \{X^{min}_{k_i k_j}, \ldots, X^{max}_{k_i k_j}\}$ such that $X = n - n'$, and the absolute value of $X$ gives the number of iterations of $G^t_{sdf}$ separating the execution of $\langle k_j + n' \cdot q_j, j\rangle$ from the execution of $\langle k_i + n \cdot q_i, i\rangle$.*

2. *Conversely, for any $X \in \{X^{min}_{k_i k_j}, \ldots, X^{max}_{k_i k_j}\}$ there exists an infinite number of tuples $(n, n') \in \mathbb{N}^2$ such that $n - n' = X$ and the channel $e$ induces a dependency relation from $\langle k_i + n \cdot q_i, i\rangle$ to $\langle k_j + n' \cdot q_j, j\rangle$.*

*Proof.*

1. Let $X \in \mathbb{Z}$ such that $n - n' = X$. If the channel $e$ induces a dependency relation from $\langle k_i + n \cdot q_i, i\rangle$ to $\langle k_j + n' \cdot q_j, j\rangle$, then by lemma 4.1 we get:

$$c_e > m_0(e) + (k_i + n \cdot q_i) \cdot p_e - (k_j + n' \cdot q_j) \cdot c_e \geq max\{0, c_e - p_e\}$$

which leads to: $c_e > m_0(e) + k_i \cdot p_e - k_j \cdot c_e + n \cdot q_i \cdot p_e - n' \cdot q_j \cdot c_e \geq max\{0, c_e - p_e\}$.

Since $G^t_{sdf}$ is consistent, according to equation (3.1) we can write $q_i \cdot p_e = q_j \cdot c_e$ and thus, the inequality above becomes:

$$c_e > m_0(e) + k_i \cdot p_e - k_j \cdot c_e + (n - n') \cdot q_i \cdot p_e \geq max\{0, c_e - p_e\}$$

Now, as we assume that $n - n' = X$, we get the following inequality:

$$c_e > m_0(e) + k_i \cdot p_e - k_j \cdot c_e + X \cdot q_i \cdot p_e \geq max\{0, c_e - p_e\}$$

Since $c_e$ can be divided by $gcd_e$, the inequality above can be rewritten as:

$$c_e - gcd_e \geq m_0(e) + k_i \cdot p_e - k_j \cdot c_e + X \cdot q_i \cdot p_e \geq max\{0, c_e - p_e\}$$

which is equivalent to:

$$\frac{c_e - gcd_e - m_0(e) - k_i \cdot p_e + k_j \cdot c_e}{q_i \cdot p_e} \geq X \geq \frac{max\{0, c_e - p_e\} - m_0(e) - k_i \cdot p_e + k_j \cdot c_e}{q_i \cdot p_e}$$

Since $X$ is an integer value, we can write:

$$\left\lfloor \frac{c_e - gcd_e - m_0(e) - k_i \cdot p_e + k_j \cdot c_e}{q_i \cdot p_e} \right\rfloor \geq X \geq \left\lceil \frac{max\{0, c_e - p_e\} - m_0(e) - k_i \cdot p_e + k_j \cdot c_e}{q_i \cdot p_e} \right\rceil$$

which leads to: $X^{max}_{k_i k_j} \geq X \geq X^{min}_{k_i k_j}$.

2. Conversely, let us consider a couple $(x, y) \in \mathbb{Z}^2$ such that:

$$x + y = 1 \tag{4.4}$$

and let $n = X \cdot x + z \cdot q_j \cdot c_e$ and $n' = -X \cdot y + z \cdot q_i \cdot p_e$. For any $X \in \{X^{min}_{k_i k_j} \ldots X^{max}_{k_i k_j}\}$ and any large positive integer $z$, there exists an infinite number of couples $(n, n') \in \mathbb{N}^2$ such that:

$$n - n' = X \cdot x + z \cdot q_j \cdot c_e + X \cdot y - z \cdot q_i \cdot p_e$$

Since $G^t_{sdf}$ is consistent, according to equation (3.1) we can write $q_i \cdot p_e = q_j \cdot c_e$ and thus, the equality above can be simplified and rewritten as:

$$n - n' = (x + y) \cdot X$$

this latter equality can further be simplified using equation (4.4) and written as:

$$n - n' = X$$

which implies the existence of a dependency relation from the firings $\langle k_i + n \cdot q_i, i\rangle$ to $\langle k_j + n' \cdot q_j, j\rangle$ and thus achieves the proof ■

### 4.2.2 A necessary and sufficient condition for admissibility

Previously, we have shown that lemma 4.2 can be used to identify the dependency relations induced by every channel of $G^t_{sdf}$ in a SWP schedule. Using this lemma, we can now characterize the set of admissible SWP schedules for $G^t_{sdf}$. For this purpose, we have established the following theorem, which stands as necessary condition and sufficient condition for admissible SWP schedules of $G^t_{sdf}$.

**Theorem 4.1.** *A SWP schedule $\sigma$ with period $\lambda$ is said admissible for $G^t_{sdf}$ if and only if for any channel $e = (i, j) \in E$ the following set of precedence constraints is fulfilled by the schedule $\sigma$:*

$$\sigma(k_j, j) - \sigma(k_i, i) \geq X^{max}_{k_i k_j} \cdot \lambda + \delta_i \qquad \forall k_i \in [0, q_i), \forall k_j \in [0, q_j),\ \forall X^{max}_{k_i k_j} \geq X^{min}_{k_i k_j}$$

*Proof.* Let us assume that a channel $e = (i, j) \in E$ induces a dependency relation from $\langle k_i + n \cdot q_i, i\rangle$ to $\langle k_j + n' \cdot q_j, j\rangle$ where $k_i \in [0, q_i)$, $k_j \in [0, q_j)$ and $n, n' \in \mathbb{N}$. If we assume that $\sigma$ is an admissible SWP schedule of $G^t_{sdf}$ then by definition. 3.2 and definition. 3.5, we get the following precedence constraint between $\langle k_i + n \cdot q_i, i\rangle$ to $\langle k_j + n' \cdot q_j, j\rangle$:

$$\sigma(k_j, j) - \sigma(k_i, i) \geq \ (n - n') \cdot \lambda + \delta_i \tag{4.5}$$

By lemma 4.2, there exists $X \in \{X^{min}_{k_i k_j}, \ldots, X^{max}_{k_i k_j}\}$ such that $n' = n - X$. Using this equality in equation (4.5), we get:

$$\sigma(k_j, j) - \sigma(k_i, i) \geq X \cdot \lambda + \delta_i \tag{4.6}$$

Now, the right part of this inequality increases with $X$ and according to lemma 4.2, there exists $(n, n') \in \mathbb{N}^2$ such that for any $X \in \{X^{min}_{k_i k_j}, \ldots, X^{max}_{k_i k_j}\}$, the channel $e$ induces a dependency relation from $\langle k_i + n \cdot q_i, i\rangle$ to $\langle k_j + n' \cdot q_j, j\rangle$. Thus, for $X = X^{max}_{k_i k_j}$ the dependency relation from $\langle k_i + n \cdot q_i, i\rangle$ to $\langle k_j + n' \cdot q_j, j\rangle$ holds if and only if:

$$\sigma(k_j, j) - \sigma(k_i, i) \geq X^{max}_{k_i k_j} \cdot \lambda + \delta_i, \quad \forall k_i \in [0, q_i), \forall k_j \in [0, q_j)$$

Conversely, if we assume this latter inequality, then by lemma 4.2, for any tuples $(k_i + n \cdot q_i, k_j + n' \cdot q_j)$ with $k_i \in [0, q_i)$, $k_j \in [0, q_j)$, $n, n' \in \mathbb{N}$ and any $X^{max}_{k_i k_j} \geq X^{min}_{k_i k_j}$, we can write $n - n' = X^{max}_{k_i k_j}$ . Using this equality, the inequality above becomes:

$$\sigma(k_j, j) - \sigma(k_i, i) \geq (n - n') \cdot \lambda + \delta_i \tag{4.7}$$

which is equivalent to:

$$\sigma(k_j, j) + n' \cdot \lambda \geq \sigma(k_i, i) + n \cdot \lambda + \delta_i \tag{4.8}$$

Now, using definition. 3.5, equation (4.8) can be simplified and rewritten as:

$$\sigma(k_j + n' \cdot q_j, j) \geq \sigma(k_i + n \cdot q_i, i) + \delta_i \tag{4.9}$$

which implies that the schedule $\sigma$ checks the dependency relation induced by the channel $e$ from $\langle k_i + n \cdot q_i, i \rangle$ to $\langle k_j + n' \cdot q_j, j \rangle$ and thus achieves the proof. ■

## 4.3 Maximum Throughput for Timed SDF graphs

In this section, we characterize the SWP schedules that achieve maximum throughput for timed SDF graphs. For this purpose, let us consider a consistent and live timed SDF graph $G_{sdf} = (V, E, P, C, M_0, \delta)$. Since the schedules that maximize throughput for $G^t_{sdf}$ minimize implicitly the iteration period (refer to section 3.3), we can compute the SWP schedules $\sigma$ that achieve maximum throughput for $G^t_{sdf}$ with the following linear programming model.

$$(P_1) \begin{cases} min \quad \lambda & \\ \sigma(k_j, j) - \sigma(k_i, i) \geq X^{max}_{k_i k_j} \cdot \lambda + \delta_i \quad \forall e = (i, j) \in E, & \\ \qquad \forall k_i \in [0, q_i), \forall k_j \in [0, q_j), \forall X^{max}_{k_i k_j} \geq X^{min}_{k_i k_j} & (1) \\ \sigma(k_i, i) \in \mathbb{Q}^+ \quad \forall i \in V, \ \forall k_i \in [0, q_i) & (2) \\ \lambda \in \mathbb{Q}^{+*} & (3) \end{cases}$$

Constraint (1) is actually a set of precedence constraints that must be fulfilled by any admissible SWP schedule $\sigma$ with period $\lambda$ for $G^t_{sdf}$. These constraints are derived from theorem 4.1 which characterizes the set of admissible SWP schedules for timed SDF graphs. Constraints (2) and (3) are integrity constraints on the decisions variables of the model ($P_1$). The number of variables in the model ($P_1$) is given by $V_k = 1 + \sum_{\forall i \in V} q_i$ and the number of

constraints $C_k$ is bounded on the upper side by $\sum_{\forall e=(i,j)\in E} q_i \cdot q_j$, where $q_i$ is the granularity of an actor $i$. Since the computation of $V_k$ and $C_k$ depends on the granularity of actors of $G^t_{sdf}$, the time and space complexity to solve the model ($P_1$) depends on the size of the repetition vector $q$ of $G^t_{sdf}$ and the size of each component of this vector.

For any timed SDF graph, the model ($P_1$) can be instantiated and solved with a linear programming solver such as CPLEX [1]. Considering the timed SDF graph depicted in figure 3.1, the instantiation of ($P_1$) gives the following system of equations:

$$
\begin{cases}
min \quad \lambda & \\
\text{e=(A,A):} & \sigma(1,A) - \sigma(0,A) \geq 3 \\
 & \sigma(0,A) - \sigma(1,A) \geq -\lambda + 3 \\
\text{e=(A,B):} & \sigma(0,B) - \sigma(0,A) \geq 3 \\
 & \sigma(0,B) - \sigma(1,A) \geq 3 \\
\text{e=(A,C):} & \sigma(0,C) - \sigma(0,A) \geq 3 \\
 & \sigma(1,C) - \sigma(0,A) \geq 3 \\
 & \sigma(2,C) - \sigma(1,A) \geq 3 \\
 & \sigma(3,C) - \sigma(1,A) \geq \lambda + 3 \\
\text{e=(B,D):} & \sigma(0,D) - \sigma(0,B) \geq 4 \\
 & \sigma(1,D) - \sigma(0,B) \geq 4 \\
\text{e=(C,D):} & \sigma(0,D) - \sigma(0,C) \geq 2 \\
 & \sigma(0,D) - \sigma(1,C) \geq 2 \\
 & \sigma(1,D) - \sigma(2,C) \geq 2 \\
 & \sigma(1,D) - \sigma(3,C) \geq 2 \\
\text{e=(D,D):} & \sigma(1,D) - \sigma(0,D) \geq 2 \\
 & \sigma(0,D) - \sigma(1,D) \geq -\lambda + 2 \\
\text{e=(D,A):} & \sigma(1,A) - \sigma(0,D) \geq -\lambda + 2 \\
 & \sigma(0,A) - \sigma(1,D) \geq -2\lambda + 2 \\
\sigma(k_i,i) \in \mathbb{Q}^+ \quad \forall i \in V,\ \forall k_i \in [0,q_i) & \\
\lambda \in \mathbb{Q}^{+*} &
\end{cases}
$$

The solution obtained with CPLEX for this system of equations is given by $\lambda^* = 9$ and thus, the maximum throughput achievable by the SDF graph is given by $\beta^* = \frac{1}{9}$. Now, using the values of $\lambda^*$ and $\sigma(k_i,i)^*$, we can describe the SWP schedules that achieve maximum throughput for the SDF graph. Figure 3.5 depicts a SWP schedule that achieves maximum

[1] https://www.ibm.com/analytics/cplex-optimizer

throughput for the SDF graph with:

$$\begin{array}{lll} \sigma(0,A)^* = 0 & \sigma(0,C)^* = 3 & \sigma(3,C)^* = 6 \\ \sigma(1,A)^* = 3 & \sigma(1,C)^* = 3 & \sigma(0,D)^* = 10 \\ \sigma(0,B)^* = 6 & \sigma(2,C)^* = 6 & \sigma(1,D)^* = 12 \end{array}$$

## 4.4 Minimum Latency for Timed SDF graphs

In this section, we propose a linear programming model to compute the admissible SWP schedules that achieve minimum latency for timed SDF graph. Let $G^t_{sdf} = (V, E, P, C, M_0, \delta)$ be a consistent and live timed SDF graph and let $\mathcal{T}$ be the latency of $G^t_{sdf}$. Since the minimum latency achievable by $G^t_{sdf}$ depends on the minimum latency achievable by every channel in $G^t_{sdf}$, we first need to characterize the latency induced by a SDF channel before formulating our linear programming model. For this purpose, we have established the following theorem.

**Theorem 4.2.** *Let $\sigma$ be an admissible SWP schedule of $G^t_{sdf}$, let $e = (i, j) \in E$ be a channel of $G^t_{sdf}$ and let $(k_i + n \cdot q_i, k_j + n' \cdot q_j)$ be a tuple of positive integers, where $k_i \in [0, q_i)$, $k_j \in [0, q_j)$ and $n, n' \in \mathbb{N}$. If the channel $e$ induces a dependency relation from the firings $\langle k_i + n \cdot q_i, i\rangle$ to $\langle k_j + n' \cdot q_j, j\rangle$, then the latency between these firings is given by:*

$$\mathcal{T}_{k_i^n k_j^{n'}} = \sigma(k_j, j) - \sigma(k_i, i) - X \cdot \lambda - \delta_i, \quad \textit{where} \quad X \in \{X^{min}_{k_i k_j}, \ldots, X^{max}_{k_i k_j}\}$$

*Proof.* Let $e = (i, j) \in E$ be a channel that induces a dependency relation from $\langle k_i + n \cdot q_i, i\rangle$ to $\langle k_j + n' \cdot q_j, j\rangle$ in the schedule $\sigma$. By equation (3.7) and definition 3.5, the latency induced by the channel $e$ is given by:

$$\mathcal{T}_{k_i^n k_j^{n'}} = \sigma(k_j, j) - \sigma(k_i, i) + (n' - n) \cdot \lambda - \delta_i \tag{4.10}$$

By lemma 4.2, there exists $X \in \{X^{min}_{k_i k_j}, \ldots, X^{max}_{k_i k_j}\}$ such that $n' = n - X$. Using this equality in equation (4.10), we get:

$$\mathcal{T}_{k_i^n k_j^{n'}} = \sigma(k_j, j) - \sigma(k_i, i) - X \cdot \lambda - \delta_i$$

and the proof is thus achieved. ∎

Using theorem 4.2, we have derived the following corollary that characterizes the mini-

mum latency achievable by a SDF channel.

**Corollary 4.1.** *Let $e = (i, j) \in E$ be a channel of $G^t_{sdf}$, let $\sigma$ be an admissible SWP schedule of $G^t_{sdf}$ and let $(k_i + n \cdot q_i, k_j + n' \cdot q_j)$ be a tuple of positive integers, where $k_i \in [0, q_i)$, $k_j \in [0, q_j)$ and $n, n' \in \mathbb{N}$. If the channel $e$ induces a dependency relation from a firing $\langle k_i + n \cdot q_i, i\rangle$ to a firing $\langle k_j + n' \cdot q_j, j\rangle$, then the minimum latency between these firings is given by:*

$$\mathcal{T}^{min}_{k_i k_j} = \sigma(k_j, j) - \sigma(k_i, i) - X^{max}_{k_i k_j} \cdot \lambda - \delta_i$$

*Proof.* Let $e = (i, j) \in E$ be a channel that induces a dependency relation from $\langle k_i + n \cdot q_i, i\rangle$ to $\langle k_j + n' \cdot q_j, j\rangle$. By theorem 4.2, the latency induced by the channel $e$ between these firings is given by:

$$\mathcal{T}_{k_i^n k_j^{n'}} = \sigma(k_j, j) - \sigma(k_i, i) - X \cdot \lambda - \delta_i$$

Now, according to lemma 4.2, if the channel $e$ induces a dependency relation from $\langle k_i + n \cdot q_i, i\rangle$ to $\langle k_j + n' \cdot q_j, j\rangle$, there exists $X \in \{X^{min}_{k_i k_j}, \ldots, X^{max}_{k_i k_j}\}$ such that $n - n' = X$. Thus, $X = X^{max}_{k_i k_j}$, gives the minimum latency $\mathcal{T}^{min}_{k_i k_j}$ induced by the channel $e$ between the firings $\langle k_i + n \cdot q_i, i\rangle$ and $\langle k_j + n' \cdot q_j, j\rangle$ and we can write:

$$\mathcal{T}^{min}_{k_i k_j} = \sigma(k_j, j) - \sigma(k_i, i) - X^{max}_{k_i k_j} \cdot \lambda - \delta_i \qquad \blacksquare$$

Using theorem 4.1 and corollary 4.1, we have derived the following linear programming model to compute the admissible SWP schedules that achieve minimum latency for $G^t_{sdf}$.

$$(P_2) \begin{cases} min \quad \mathcal{T} & \\ \sigma(k_j, j) - \sigma(k_i, i) \geq X^{max}_{k_i k_j} \cdot \lambda + \delta_i, \quad \forall e = (i, j) \in E, & \\ \qquad \forall k_i \in [0, q_i), \forall k_j \in [0, q_j), \forall X^{max}_{k_i k_j} \geq X^{min}_{k_i k_j} & (1) \\ \mathcal{T} \geq \sigma(k_i, i) + \delta_i, \quad \forall i \in V, \ \forall k_i \in [0, q_i) & (2) \\ \sigma(k_i, i) \in \mathbb{Q}^+ \quad \forall i \in V \ \forall k_i \in [0, q_i), & (3) \\ \lambda \in \mathbb{Q}^{+*} & (4) \end{cases}$$

Constraint (1) characterizes the set of precedence constraints that must be fulfilled by any admissible schedule of $G^t_{sdf}$. This constraint also ensures that the minimum latency induced by every channel of $G^t_{sdf}$ between two firings is greater or equal to 0. Indeed, according to corollary 4.1, if a channel $e = (i, j) \in E$ induces a dependency relation from a

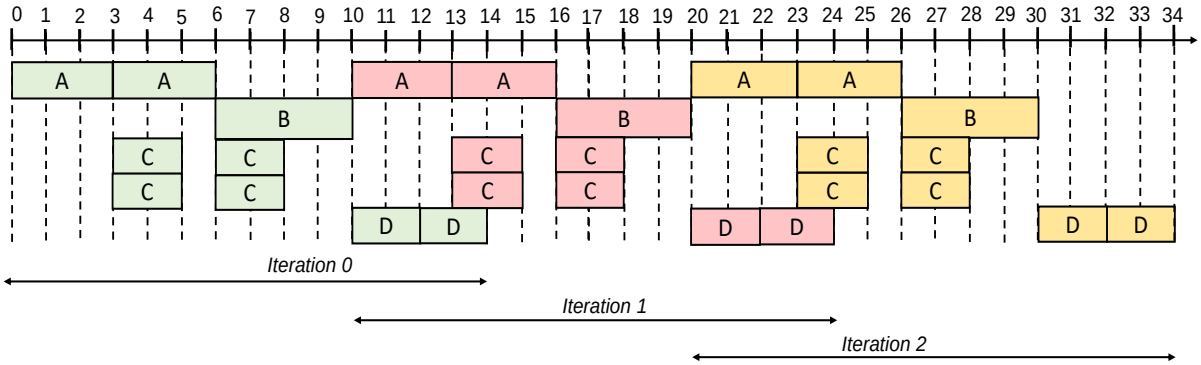



Figure 4.1: A SWP Schedule of period $\lambda = 10$ and latency $\mathcal{T}^* = 14$ for the timed SDF graph depicted in figure 3.1.

firing $\langle k_i + n \cdot q_i, i \rangle$ to a firing $\langle k_j + n' \cdot q_j, j \rangle$ then the minimum latency between these firings is given by $\mathcal{T}^{min}_{k_i k_j} = \sigma(k_j, j) - \sigma(k_i, i) - X^{max}_{k_i k_j} \cdot \lambda - \delta_i$. Since latency is a positive value, for every channel that induces a dependency relation from $\langle k_i + n \cdot q_i, i \rangle$ to a firing $\langle k_j + n' \cdot q_j, j \rangle$ we can write $\mathcal{T}^{min}_{k_i k_j} \geq 0$, which leads to constraint (1). Constraint (2) ensures that the latency of $G^t_{sdf}$ is greater or equal to the finishing time of every firing of any actor within a single iteration of $G^t_{sdf}$. Constraints (3) and (4) are integrity constraints on the decisions variables of the model ($P_2$). The number of variables in an instance of ($P_2$) is given by $V_k = 2 + \sum_{\forall i \in V} q_i$ and the number of constraints $C_k$ is bounded on the upper side by $\sum_{\forall e=(i,j) \in E} q_i \cdot q_j + \sum_{\forall i \in V} q_i$, where $q_i$ is the granularity of an actor $i$.

For any timed SDF graph, the model ($P_2$) can be instantiated and solved with CPLEX. Like the linear programming model ($P_1$), it is important to mention that the time and space complexity to solve ($P_2$) for a timed SDF graph depends both on the size of the repetition vector of this graph and the size of the components of this vector.

Figure 4.1 depicts a SWP schedule that gives minimum latency for the timed SDF graph of our running example (refer to figure 3.1). In this schedule, the minimum latency achievable by the SDF graph is equal to $\mathcal{T}^* = 14$. Although this schedule achieves minimum latency for the graph, one can note that it does not achieve maximum throughput since throughput $\beta = \frac{1}{10}$ achievable by this schedule is suboptimal compared to the throughput $\beta^* = \frac{1}{9}$ achievable by the schedule depicted in figure 3.7. This shows that the schedules that achieve minimum latency for a timed SDF graph do not necessarily achieve maximum throughput.

## 4.5 Conclusion

In this chapter, we have characterized admissible SWP schedules for timed SDF graphs and we have proposed two linear programming models ($P_1$) and ($P_2$) which enable respectively to compute the SWP schedules that achieve maximum throughput and minimum latency for timed SDF graphs. We will note that these models consider only the precedence constraints of timed SDF graphs and they do accommodate neither resource nor communication constraints. In the next chapter, we will consider both precedence, resource, and communication constraints to compute SWP schedule of SDF graphs on heterogeneous multiprocessor architectures.

# CHAPTER 5

# Software Pipelined Scheduling under Resources and Communication Constraints

## Contents

## 5.1 Introduction

In this chapter, we study the software pipelined (SWP) scheduling problem of SDF graphs on heterogeneous multiprocessor architectures (HMAs) under resource and communication constraints with the goal of optimizing throughput. The chapter is structured into two parts. In the first part, we present our integer linear programming (ILP) model for an exact resolution of the scheduling problem. In the second part, we present a decomposed SWP scheduling heuristic build upon the heuristic of Gasperoni and Schwielgelshohn, which provides approximated solutions for the scheduling problem.

## 5.2 An Integer Linear Programming Model

In this section, we present an ILP formulation for the problem described in section 3.4. Our ILP formulation consists of different constraints separated into resource, communication, cyclicity and precedence constraints. The inputs of the ILP model are a SDF graph $G_{sdf} = (V, E, P, C, M_0)$ and a HMA $G_{hma} = (R, \Delta, \Gamma)$, and the scheduling entities are the firings of actors belonging to $G_{sdf}$. It is important to recall that the SDF graphs considered in this thesis consist of stateful and/or stateless actors. The objective of this ILP formulation is to minimize the period $\lambda$ (or conversely to maximize the throughput $\beta$) achievable by $G_{sdf}$ in a SWP schedule $\sigma$. Let $k_i \in \mathbb{N}$ and $q_i \in \mathbb{N}^*$ respectively denotes the firing index and the granularity of an actor $i \in V$.

### 5.2.1 Cyclicity Constraints

In order to ensure that each firing of each actor is executed cyclically according to a period $\lambda$, our ILP model incorporates the set of cyclicity constraints that must be fulfilled by any SWP schedule $\sigma$. These constraints are expressed by equation (3.4).

### 5.2.2 Resource Constraints

When scheduling $G_{sdf}$ on $G_{hma}$, we need to ensure that each firing of each actor is assigned exactly to one PU. In order to formulate these constraints, we define a $0-1$ integer variable $w_{x,k_i,i}$ such that:

$$w_{x,k_i,i} = \begin{cases} 1 & \text{if the firing } \langle k_i, i \rangle \text{ has been assigned to the PU } x. \\ 0 & \text{otherwise.} \end{cases}$$

Using this variable, we formulate the following set of resource constraints which ensure that each firing of each actor is assigned to a single PU:

$$\sum_{\forall x \in R} w_{x,k_i,i} = 1, \quad \forall i \in V, \ \forall k_i \in [0, q_i) \tag{5.1}$$

Since $G_{sdf}$ consists of stateless actors whose firings may be executed in parallel and/or stateful actors whose firings may be pipelined, we need to ensure that the execution of independent firings of actors cannot overlap on a same PU. For this purpose, we formulate the following set of inequalities:

$$\begin{cases} \sigma(k_i, i) + \Delta_{xi} - \sigma(k_j, j) \leq M(1 - w_{x,k_i,i} \cdot w_{x,k_j,j}) \\ \qquad\qquad or \\ \sigma(k_j, j) + \Delta_{xj} - \sigma(k_i, i) \leq M(1 - w_{x,k_i,i} \cdot w_{x,k_j,j}) \\ \forall x \in R, \forall i, j \in V, \forall k_i \in [0, q_i), \forall k_j \in [0, q_j) \end{cases} \tag{5.2}$$

Actually, equation (5.2) is a set of non-linear disjunctive constraints which assert that two firings assigned to the same PU cannot be executed at the same time on this PU. In these constraints we use M, a big integer value such that the constraints hold only for the firings assigned to the same computing unit, i.e $w_{x,k_i,i} = w_{x,k_j,j} = 1$. The disjunctive constraints described by equation (5.2) could be linearized in two steps. First, we replace $M(1 - w_{x,k_i,i} \cdot w_{x,k_j,j})$ by $M(2 - w_{x,k_i,i} - w_{x,k_j,j})$ and then equation (5.2) becomes:

$$\begin{cases} \sigma(k_i, i) + \Delta_{xi} - \sigma(k_j, j) \leq M(2 - w_{x,k_i,i} - w_{x,k_j,j}) \\ \qquad\qquad or \\ \sigma(k_j, j) + \Delta_{xj} - \sigma(k_i, i) \leq M(2 - w_{x,k_i,i} - w_{x,k_j,j}) \\ \forall x \in R, \forall i, j \in V, \forall k_i \in [0, q_i), \forall k_j \in [0, q_j) \end{cases} \tag{5.3}$$

Second, we introduce another $0-1$ integer variable $d_{k_i,i,k_j,j}$ such that:

$$d_{k_i,i,k_j,j} = \begin{cases} 1 & \text{if the firing } \langle k_i, i\rangle \text{ is scheduled before the firing } \langle k_j, j\rangle. \\ 0 & \text{otherwise.} \end{cases}$$

Now, using the variable $d_{k_i,i,k_j,j}$ the set of disjunctive constraints described by equation (5.3) could be linearized and rewritten as:

$$\begin{cases} \sigma(k_i, i) + \Delta_{xi} - \sigma(k_j, j) \leq M(2 - w_{x,k_i,i} - w_{x,k_j,j}) + M(1 - d_{k_i,i,k_j,j}) \\ \sigma(k_j, j) + \Delta_{xj} - \sigma(k_i, i) \leq M(2 - w_{x,k_i,i} - w_{x,k_j,j}) + M d_{k_i,i,k_j,j} \\ \forall x \in R, \forall i, j \in V, \forall k_i \in [0, q_i), \forall k_j \in [0, q_j) \end{cases} \tag{5.4}$$

Simplifying further, equation (5.4) becomes:

$$\begin{cases} \sigma(k_i, i) + \Delta_{xi} - \sigma(k_j, j) \leq M(3 - w_{x,k_i,i} - w_{x,k_j,j} - d_{k_i,i,k_j,j}) \\ \sigma(k_j, j) + \Delta_{xj} - \sigma(k_i, i) \leq M(2 - w_{x,k_i,i} - w_{x,k_j,j} + d_{k_i,i,k_j,j}) \\ \forall x \in R, \forall i, j \in V, \forall k_i \in [0, q_i), \forall k_j \in [0, q_j) \end{cases} \tag{5.5}$$

Equation (5.5) is actually a set of linear constraints that enforces a ILP solver to schedule the firings $\langle k_i, i \rangle$ before $\langle k_j, j \rangle$ when the both firings are assigned to the same PU. Otherwise, it enforces the firings $\langle k_j, j \rangle$ to be scheduled before $\langle k_i, i \rangle$ on the PU.

### 5.2.3 Communication and Precedence Constraints

In order to ensure that our ILP formulation can generate admissible SWP schedules, we should incorporate the precedence constraints induced by every channel of $G_{sdf}$. By Theorem 4.1, a SWP pipelined schedule $\sigma$ is said admissible for a timed SDF graph $G^t_{sdf}$, if and only if for every channel $e = (i, j)$ of $G^t_{sdf}$, the following precedence constraints are fulfilled:

$$\sigma(k_j, j) - \sigma(k_i, i) \geq X^{max}_{k_i k_j} \cdot \lambda + \delta_i \qquad \forall k_i \in [0, q_i), \forall k_j \in [0, q_j),\ \forall X^{max}_{k_i k_j} \geq X^{min}_{k_i k_j} \tag{5.6}$$

Actually, the constraints expressed by equation (5.6) were constructed for timed SDF graphs regardless of communication constraints of a HMA and by assuming that the worst case cost to execute a single firing of any actor is equal to $\delta_i$. However, regarding the description of $G_{hma}$ (refer to section 3.4), the execution cost of a single firing of every actor depends on the PU on which this firing is assigned. Moreover, whether a channel $e = (i, j)$ of $G_{sdf}$ induces a dependency relation between two firings $\langle k_i, i \rangle$ and $\langle k_j, j \rangle$, which are assigned respectively to a PU $x \in R$ and a PU $y \in R$ (with $x \neq y$), there exists a non-zero communication cost between these firings. In order to consider these different costs in our ILP formulation, we reformulate the precedence constraints described by equation (5.6) and we get the following set of precedence and communication constraints:

$$\begin{aligned} \sigma(k_j, j) - \sigma(k_i, i) \geq X^{max}_{k_i k_j} \cdot \lambda + \sum_{\forall x \in R} \Delta_{xi} \cdot w_{x,k_i,i} + \sum_{\forall x \in R} \sum_{\forall y \in R} \Gamma_{xy} \cdot w_{x,k_i,i} \cdot w_{y,k_j,j}, \\ \forall e = (i, j) \in E, \forall k_i \in [0, q_i), \forall k_j \in [0, q_j), \forall X^{max}_{k_i k_j} \geq X^{min}_{k_i k_j} \end{aligned} \tag{5.7}$$

Actually, equation (5.7) is a set of non-linear constraints, each associated with a pair of dependent firings. These constraints could be linearized and rewritten as follows:

$$\begin{aligned}\sigma(k_j, j) - \sigma(k_i, i) \geq X^{max}_{k_i k_j} \cdot \lambda + \Delta_{xi} \cdot w_{x,k_i,i} + \Gamma_{xy} \cdot (w_{x,k_i,i} + w_{y,k_j,j} - 1),\\ \forall x, y \in R, \forall e = (i, j) \in E, \forall k_i \in [0, q_i), \forall k_j \in [0, q_j), \forall X^{max}_{k_i k_j} \geq X^{min}_{k_i k_j}\end{aligned} \tag{5.8}$$

In order to justify the equivalence between the constraints described by equations (5.7) and (5.8), let us consider a channel $e = (i, j) \in E$ and let $\langle k_i, i\rangle$ and $\langle k_j, j\rangle$ be two firings such that the execution of $\langle k_i, i\rangle$ precedes the execution of $\langle k_j, j\rangle$. Since each firing is assigned exactly to a single PU (i.e. $\sum_{\forall x \in R} w_{x,k_i,i} = \sum_{\forall x \in R} w_{x,k_j,j} = 1$), the two sums $u = \sum_{\forall x \in R} \Delta_{xi} \cdot w_{x,k_i,i}$ and $v = \sum_{\forall x \in R} \sum_{\forall y \in R} \Gamma_{xy} \cdot w_{x,k_i,i} \cdot w_{x,k_i,i}$ contain only one term different from zero. In fact, if $\langle k_i, i\rangle$ is assigned to $x^* \in R$ and $\langle k_j, j\rangle$ is assigned to $y^* \in R$, we can write $w_{x^*,k_i,i} = w_{y^*,k_j,j} = 1$, $u = \Delta_{x^* i}$, $v = \Gamma_{x^* y^*} = \Gamma_{x^* y^*}(w_{x^*,k_i,i} + w_{x^*,k_j,j} - 1)$ and thus, the constraints described by equations (5.7) and (5.8) can be rewritten as follows:

$$\sigma(k_j, j) - \sigma(k_i, i) \geq X^{max}_{k_i k_j} \cdot \lambda + u + v \tag{5.9}$$

which proves the equivalence between equations (5.7) and (5.8).

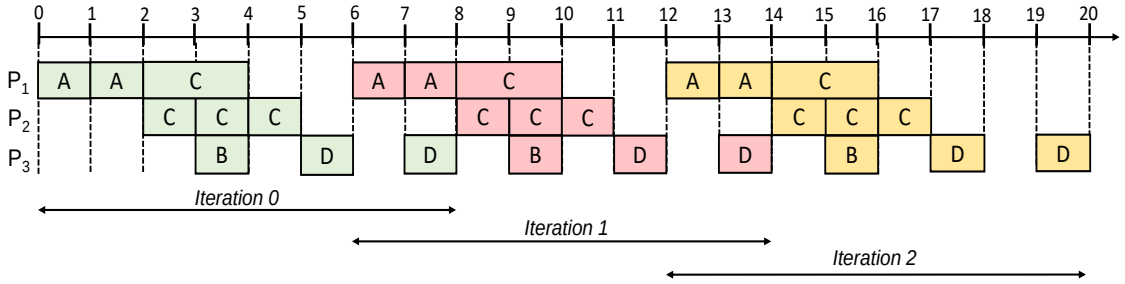



Figure 5.1: An optimal scheduling solution obtained for the non-timed SDF graph and the architecture of our running example.

To summarize, our ILP model incorporates equations (3.4), (5.1), (5.5) and (5.8) as constraints and $\lambda$ as objective function. A compact description of the model is given by:

$$
(P_3)\begin{cases}
min \quad \lambda & \\
\sigma(k_i + n \cdot q_i, i) = \sigma(k_i, i) + n \times \lambda, \quad \forall i \in V,\ \forall k_i \in [0, q_i),\ \forall n \in \mathbb{N}. & (3.4)\\
\sum\limits_{\forall x \in R} w_{x,k_i,i} = 1, \quad \forall i \in V,\ \forall k_i \in [0, q_i). & (5.1)\\
\sigma(k_i, i) + \Delta_{xi} - \sigma(k_j, j) \leq M(3 - w_{x,k_i,i} - w_{x,k_j,j} - d_{k_i,i,k_j,j}), \quad \forall x \in R, & \\
\qquad \forall i, j \in V, \forall k_i \in [0, q_i), \forall k_j \in [0, q_j). & (5.5.1)\\
\sigma(k_j, j) + \Delta_{xj} - \sigma(k_i, i) \leq M(2 - w_{x,k_i,i} - w_{x,k_j,j} + d_{k_i,i,k_j,j}), \quad \forall x \in R, & \\
\qquad \forall i, j \in V, \forall k_i \in [0, q_i), \forall k_j \in [0, q_j). & (5.5.2)\\
\sigma(k_j, j) - \sigma(k_i, i) \geq X^{max}_{k_i k_j} \cdot \lambda + \Delta_{xi} \cdot w_{x,k_i,i} + \Gamma_{xy} \cdot (w_{x,k_i,i} + w_{y,k_j,j} - 1), & \\
\qquad \forall x, y \in R, \forall e = (i, j) \in E, \forall k_i \in [0, q_i), \forall k_j \in [0, q_j), \forall X^{max}_{k_i k_j} \geq X^{min}_{k_i k_j}. & (5.8)\\
w_{x,k_i,i} \in \{0, 1\}, \quad \forall x \in R, \forall i \in V, \forall k_i \in [0, q_i). & (I_1)\\
\sigma(k_i, i) \in \mathbb{Q}^+ \quad \forall i \in V,\ \forall k_i \in [0, q_i). & (I_2)\\
\lambda \in \mathbb{Q}^{+*}. & (I_3)
\end{cases}
$$

where constraints $(I_1)$, $(I_2)$ and $(I_3)$ are integrity constraints on the decision variables of the model $(P_3)$. For any SDF graph $G_{sdf}$ and any architecture $G_{hma}$, the ILP model $(P_3)$ can be instantiated and solved with the ILP solver of CPLEX. Let us consider the non-timed SDF graph shown in figure 2.6b and the architecture on which this graph is intended to be deployed (figure 3.8). An optimal scheduling solution obtained with our ILP formulation for this graph is depicted in figure 5.1, where $\lambda^* = 6$ and:

$$
\begin{array}{lll}
\sigma(0, A)^* = 0 & \sigma(0, C)^* = 2 & \sigma(3, C)^* = 4\\
\sigma(1, A)^* = 1 & \sigma(1, C)^* = 2 & \sigma(0, D)^* = 5\\
\sigma(0, B)^* = 3 & \sigma(2, C)^* = 3 & \sigma(1, D)^* = 7
\end{array}
$$

In this schedule, one can note that each firing of each actor is executed cyclically according to the period $\lambda^*$ and data parallelism is exploited since some firings of the stateless actor $C$ can execute out of order on different PUs. Although the ILP solver of CPLEX can find a schedule that optimizes the value of $\lambda$, the time to find this schedule can be exponential for some instances of SDF graphs and architecture models. Therefore, we should design approximated techniques to solve efficiently the problem. For this purpose we focus on decomposed SWP scheduling approaches. Decomposed SWP scheduling is a technique which consists in separating the SWP scheduling problem of a dataflow graph into two sub-problems, the first being to satisfy the precedence and cyclicity constraints of ($P_3$) the graph and the second being to satisfy the resource constraints.

## 5.3 Decomposed Software Pipelined Scheduling

In this section, we present a heuristic denoted by heterogeneous cyclic scheduling (HCS) that is designed to solve the scheduling problem described by the ILP model ($P_3$). HCS is a decomposed SWP scheduling heuristic based upon the heuristic of Gasperoni and Schwielgelshohn [48], one of the first decomposed SWP scheduling heuristic for executing cyclic dataflow graphs under resource constraints. In the first subsection, we briefly describe the heuristic of Gasperoni and Schwielgelshohn that we denote by GS, and in the second subsection, we present in detail our heuristic.

### 5.3.1 GS Heuristic

**Description.** Let $G^t_{hsdf} = (V, E, M_0, \delta)$ be a timed HSDF graph —i.e. a timed SDF graph, where the production and consumption rates of actors are all equal to 1 —and let us consider a multiprocessor architecture with $p$ identical PUs for $G^t_{hsdf}$, where $p$ is a finite number ($p \neq \infty$) and the inter-PU communication costs are negligible. The main idea of GS is the following. Assume that we have an optimal SWP schedule $\sigma_\infty$ of period $\lambda_\infty$ of $G^t_{hsdf}$ for unlimited resources —i.e. a SWP schedule with $p = \infty$ —and that we want to deduce a SWP schedule $\sigma$ of period $\lambda$ of $G^t_{hsdf}$ under resource constraints —i.e. a SWP schedule with $p \neq \infty$ —. A way of building $\sigma$ is to keep the structure of $\sigma_\infty$ and to reorganize the execution of actors within this latter schedule in such a way as to find the period $\lambda$ that meets both resource and precedence constraints. In order to achieve this, GS proceeds in four steps. Each of these steps is described by Algorithm 2:

**Algorithm 2:** GS Heuristic

**Input:** $G^t_{hsdf} = (V, E, M_0, \delta)$, an architecture with $p$ identical PUs.

**Output:** A valid SWP schedule $\sigma$ of $G^t_{hsdf}$ over T iterations.

*// Step 1: Scheduling of $G^t_{hsdf}$ without resource constraints.*

Compute an optimal SWP schedule $\sigma_\infty$ of period $\lambda_\infty$ for $G^t_{hsdf}$ under unlimited number of PUs and let $\sigma_\infty(n, i)$ be the starting time of the $n^{th}$ firing of actor $i \in V$ in $\sigma_\infty$;

*// Step 2: Construction of an acyclic dependency graph $G_{adg}$*

$E_{adg} \leftarrow E$;

**foreach** *channel* $e = (i, j) \in E_{adg}$ **do**

  **if** $(\sigma_\infty(n, j) \bmod \lambda_\infty < \sigma_\infty(n, i) \bmod \lambda_\infty + \delta_i)$ **then**

    delete $e$ from $E_{adg}$;

  **end**

**end**

Set $G_{adg} = (V, E_{adg}, \delta)$;

*// Step 3: Scheduling of $G_{adg}$ under resource constraints.*

Find a list schedule $\sigma_a$ of $G_{adg}$ under $p$ identical PUs and let $\lambda$ be the length of this schedule $\sigma_a$ and $\sigma_a(i)$ be the starting time of any node $i \in V_{adg}$ in the schedule $\sigma_a$;

*// Step 4: Computation of a SWP schedule $\sigma$ of $G^t_{hsdf}$ under resource constraints.*

**foreach** $i \in V$ **do**

  **for** $n \leq T$ **do**

    $$\sigma(n, i) = \sigma_a(i) + \left(n + \left\lfloor \frac{\sigma_\infty(n, i)}{\lambda_\infty} \right\rfloor\right) \cdot \lambda$$

  **end**

**end**

return $\sigma$;

- **Step 1.** A SWP schedule $\sigma_\infty$ of period $\lambda_\infty$ is built for $G^t_{hsdf}$ with an infinite number of resources. Since this schedule is not constrained by the number of resources, it can be constructed in polynomial-time using an algorithmic approach [48] or by instantiating and solving the linear programming model ($P_1$).

- **Step 2.** The dependency information described by the repetitive patterns of the schedule $\sigma_\infty$ are used to construct an acyclic dependency graph $G_{adg} = (V, E_{adg}, \delta)$,

where $E_{adg}$ is a set that contains every channel $e = (i, j) \in E$, except the channels for which the following inequality is checked:

$$\sigma_\infty(n, i) \bmod \lambda_\infty + \delta_i > \sigma_\infty(n, j) \bmod \lambda_\infty, \quad \forall n \in \mathbb{N} \tag{5.10}$$

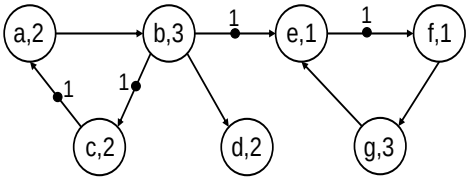
where $\sigma_\infty(n, j) \bmod \lambda_\infty$ and $\sigma_\infty(n, j) \bmod \lambda_\infty$ respectively are the times at which nodes $i$ and node $j$ start for the first time in the schedule $\sigma_\infty$. Actually, the inequality above holds only for loop-carried dependency channels (i.e a channel with tokens) between the firings of any pair of actors. Consequently, every channel $e = (i, j) \in E_{adg}$ is a direct dependency channel (i.e a channel without tokens).

- **Step 3.** A list-scheduling of $G_{adg}$ is performed under resource constraints to determine a schedule $\sigma_a$ of length $\lambda$.
- **Step 4.** A valid SWP schedule $\sigma$ of period $\lambda$ is calculated with the following approximation equation, which ensures the respect of cyclicity constraints for every actor of $G^t_{hsdf}$ while satisfying both resource and precedence constraints.

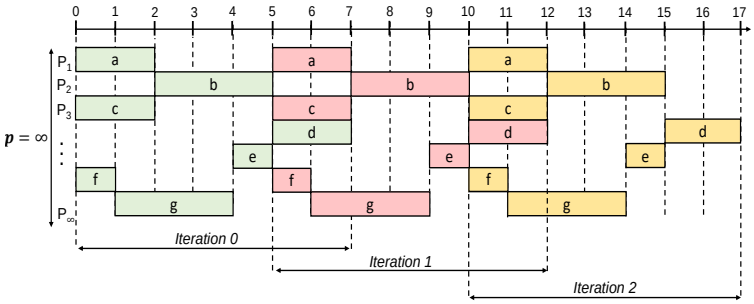
$$\sigma(n, i) = \sigma_a(i) + \left(n + \left\lfloor \frac{\sigma_\infty(n, i)}{\lambda_\infty} \right\rfloor\right) \cdot \lambda, \quad \forall i \in V, \forall n \in \mathbb{N} \tag{5.11}$$

The correctness of the heuristic *GS* can be found in [48]. To better explain the heuristic *GS*, we illustrate the different steps described above, through the following example.

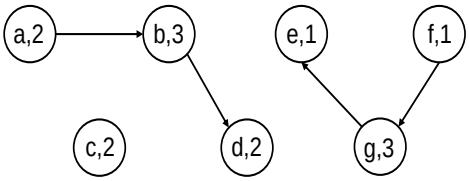
**Example.** Let $G^t_{hsdf}$ be the timed HSDF graph depicted in figure 5.2a and let us assume that we want to schedule this graph on a multiprocessor architecture with two identical PUs (i.e. $p = 2$). Applying the first step of algorithm 2, an optimal SWP schedule $\sigma_\infty$ of period $\lambda_\infty$ can be calculated for $G^t_{hsdf}$ by instantiating and solving the linear programming model ($P_1$) with CPLEX. Figure 5.2b shows a schedule $\sigma_\infty$ for $G^t_{hsdf}$, where $\lambda_\infty = 5$. Based on this schedule $\sigma_\infty$, one can derive in step 2, an acyclic dependency graph $G_{adg}$, which contains only the direct dependency channels of $G^t_{hsdf}$. Actually, the graph $G_{adg}$ is obtained by deleting all the dependency channels of $G^t_{hsdf}$ for which the condition stated in equation 5.10 holds. For instance, let us consider the channel $(c, a)$ within the graph $G^t_{hsdf}$. In each iteration of the schedule $\sigma_\infty$, one can note that the execution of $n^{th}$ firing of actor $a$ starts before the execution of the $n^{th}$ firing of actor $c$ finishes, i.e $\sigma_\infty(n, c) \bmod \lambda_\infty + \delta_c > \sigma_\infty(n, a) \bmod \lambda_\infty$. This means that the dependency relation from $a$ to $c$ is not a direct dependency but rather a loop-carried dependency; thus, the channel (c,a) can be removed from the set of channels of



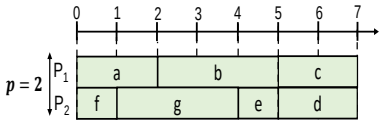

(a) A timed HSDF graph $G^t_{hsdf}$



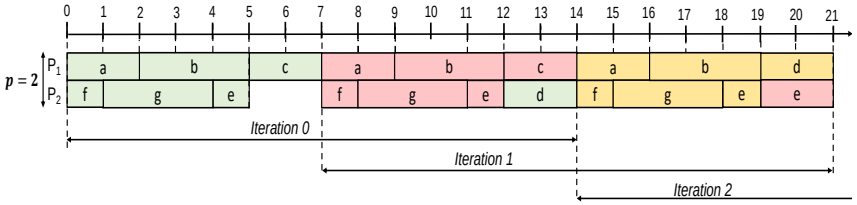

(b) An optimal SWP schedule $\sigma_\infty$ of $G^t_{hsdf}$ for $p = \infty$




(c) Acyclic precedence graph $G_{adg}$




(d) list schedule $\sigma_a$ of $G_{adg}$ for $p = 2$




(e) A SWP schedule $\sigma$ of period $\lambda$ for $G^t_{hsdf}$ under resource constraints

Figure 5.2: An illustration example for the Heuristic GS (algorithm 2).

$G_{adg}$. Figure 5.2c depicts the acyclic graph $G_{adg}$ derived from $G^t_{hsdf}$. Now, in the third step, the graph $G_{adg}$ is scheduled under resource constraints (p=2) by means of a list scheduler $\sigma_a$ that executes as soon as possible the actors of $G_{adg}$ on the $p$ identical resources. Figure 5.2d depicts this schedule whose length is $\lambda = 6$ for our example. Finally, in the fourth step, a SWP schedule $\sigma$ is calculated under resource constraints using the approximation stated by equation . For our example, this schedule is depicted in figure 5.2e. As it can be noted, the period of the schedule $\sigma$ is given by length of the schedule $\sigma_a$. Hence, the performance of GS depends on that of the schedule $\sigma_a$.

**Theoretical performance.** Gasperoni and Schwielgelshohn have given a theoretical upper bound to the iteration period $\lambda$ of the schedule returned by algorithm 2. Let $\lambda^*$ be the period of an optimal SWP schedule $\sigma$ of $G_{sdf}$ for $p$ identical PUs. The upper bound of $\lambda$

is characterized by the following inequality:

$$p \cdot \lambda \leq p \cdot \lambda^* + (p-1) \cdot \Phi$$

where $\Phi$ is the length of the longest path in $G_{adg}$. Owing to channel deletion strategy, $\Phi \leq \lambda_\infty + \delta_{max} - 1$ (see lemma 1 in [48]). Thus, the inequality above can be rewritten as:

$$p \cdot \lambda \leq p \cdot \lambda^* + (p-1)(\lambda_\infty + \delta_{max} - 1)$$

which leads to:

$$\frac{\lambda}{\lambda^*} \leq 2 - \frac{1}{p} + \left(\frac{p-1}{p}\right)\left(\frac{\lambda_\infty + \delta_{max} - 1}{\lambda^*}\right)$$

### 5.3.2 HCS Heuristic

Let $G_{sdf} = (V, E, M_0, P, C)$ be a consistent, live and non-timed SDF graph and let $G_{hma} = (R, \Delta, \Gamma)$ be the architecture on which $G_{sdf}$ is intended to be deployed. The heuristic HCS shares the same idea with the heuristic GS. However, the main difference between these heuristics is that HCS accommodates both the resource and communication constraints of heterogeneous multiprocessor architectures to schedule SDF graphs. Before presenting HCS, we first convert $G_{sdf}$ into an equivalent homogeneous SDF graph $G_{hsdf} = (V', E', M'_0)$. The construction of $G_{hsdf} = (V', E', M'_0)$ is performed using Algorithm 1, which generates for any consistent SDF graph, an equivalent homogeneous representation with a minimal number of channels. Actually, $V'$ is a set of nodes $i_k$ where $i \in V$ and $k \in [1, q_i]$, $q_i$ being the granularity of actor $i$. $E'$ is the set of arcs between these firings and $M'_0$ is a function, that associates with each arc $e' = (i_k, j_{k'}) \in E'$, an initial number of tokens. Previously presented, figure 3.2 depicts the equivalent homogeneous graph obtained with Algorithm 1 for the non-timed SDF graph of our running example (refer to figure 2.6b).

HCS takes as inputs $G_{sdf}$, $G_{hsdf}$ and $G_{hma}$, and it outputs a SWP schedule of $G_{hsdf}$ on $G_{hma}$. The heuristic is illustrated by Algorithm 3. Actually, HCS consists of four steps. Each of these steps are described below.

**Algorithm 3:** HCS-Heterogeneous Cyclic Scheduling

**Input:** $G_{sdf} = (V, E, M_0, P, C)$, $G_{hsdf} = (V', E', M'_0)$, $G_{hma} = (R, \Delta, \Gamma)$.

**Output:** A SWP schedule $\sigma$ for $G_{sdf}$ on $G_{hma}$ over T iterations.

// Step 1: Scheduling without resource and communication constraints.

Set $G_{hsdf}$ into a timed homogeneous SDF graph $G^t_{hsdf} = (V', E', M'_0, \delta)$ and find an optimal SWP schedule $\sigma_\infty$ of period $\lambda_\infty$ for $G^t_{hsdf}$ under unlimited number of resources, and let $\sigma_\infty(n, i_k)$ be the starting time of the $n^{th}$ firing of actor $i_k \in V'$ in the schedule $\sigma_\infty$;

// Step 2: Construction of an acyclic dependency graph.

$E_{adg} \leftarrow E'$;

**foreach** *channel* $e' = (i_k, j_{k'}) \in E_{adg}$ **do**

  **if** $(\sigma_\infty(n, j_{k'}) \bmod \lambda_\infty < \sigma_\infty(n, i_k) \bmod \lambda_\infty + \delta_i)$ **then**

    delete the channel $e'$ from $E_{adg}$;

  **end**

**end**

Set $G_{adg} = (V', E_{adg}, \delta)$;

// Step 3: List scheduling under resource and communication constraints.

Use Algorithm 4 to find a list schedule $\sigma_a$ of $G_{adg}$ on $G_{hma}$ ;

// Step 4: Derive a SWP schedule $\sigma$ of period $\lambda$ for $G_{hsdf}$ under the resource and communication constraints of $G_{hma}$.

$\lambda \leftarrow 0$;

**foreach** *actor* $i_k \in V'$ **do**

  Let $succ(i_k)$ be the set of successors of $i_k$ in the graph $G_{hsdf}$, $AFT(i_k)$ be the actual finishing time of $i_k$ in the schedule $\sigma_a$ and $proc(i_k)$ be the processing unit on which $i_k$ is mapped to in the schedule $\sigma_a$;

  **foreach** $j_{k'} \in succ(i_k)$ **do**

    **if** $(\lambda < AFT(i_k) + \Gamma_{proc(i_k)proc(j_{k'})})$ **then**

      $\lambda \leftarrow AFT(i_k) + \Gamma_{proc(i_k)proc(j_{k'})}$;

    **end**

  **end**

**end**

**foreach** *actor* $i_k \in V'$ **do**

  **for** $n \leq T$ **do**

    $\sigma(n, i_k) = \sigma_a(i_k) + n \cdot \lambda$

  **end**

**end**

return $\sigma$;

**Algorithm 4:** HAS-Heterogeneous Acyclic Scheduling

**Input:** $G_{adg} = (V', E_{adg}, \delta)$, $G_{hma} = (R, \Delta, \Gamma)$

**Output:** A list schedule $\sigma_a$

// Phase 1: prioritizing

Compute the scheduling rank of each node $i_k \in V'$ and generate a scheduling list, where nodes are sorted by increasing order of scheduling ranks;

// Phase 2: mapping

**while** *the scheduling list is not empty* **do**

  Select the first node $i_k$ from the scheduling list;

  **foreach** *resource* $x \in R$ **do**

    Compute $EFT(x, i_k)$ using an insertion-based scheduling policy;

  **end**

  Get the resource $x$ that minimizes the value of EFT and map the node $i_k$ on $x$;

**end**

return the schedule $\sigma_a$ ;

## Description of HCS

- **Step 1.** Firstly, the graph $G_{hsdf}$ is set into a timed homogeneous SDF graph $G^t_{hsdf} = (V', E', M'_0, \delta)$, where $\delta : V' \rightarrow \mathbb{N}^*$ is a function that associates a time budget $\delta_{i_k}$ with every node $i_k \in V'$, such that:

$$\delta_{i_k} = \Delta_i^{(max)} + \Gamma^{max} \tag{5.12}$$

  where $\Delta_i^{max}$ is the worst delay to process a firing actor $i$ on the architecture $G_{hma}$ and $\Gamma^{max}$ is the maximum inter-PU communication delay. Figure 5.3a illustrates the graph $G^t_{hsdf}$ for our running example, where for any value of $k$ the time costs of nodes are given by $\delta_{A_k}$=7, $\delta_{B_k}$=5, $\delta_{C_k}$=8, and $\delta_{D_k}$=5. Secondly, an initial SWP schedule $\sigma_\infty$ is calculated. Actually, this schedule does not satisfy neither the resource constraints nor the communication of $G_{hma}$, however it gives some interesting information about the dependency relations between the nodes of $G_{hsdf}$.

- **Step 2.** According to the dependency information given by the schedule $\sigma_\infty$, HCS constructs an acyclic dependency graph $G_{adg} = (V', E_{adg}, \delta)$, where $E_{adg}$ is the set of direct dependency channels of $G^t_{hsdf}$. Actually, the graph $G_{adg}$ is obtained by deleting every loop-carried dependency channel in $G^t_{hsdf}$. The channel deletion strategy used is the same than that previously described for the heuristic GS.

- **Step 3.** HCS performs the list-scheduling of the acyclic graph $G_{adg}$ under resource

and communication constraints. For this purpose, we have designed a list-scheduling algorithm denoted by HAS (algorithm 4), where HAS stands for heterogeneous acyclic scheduling. HAS takes as inputs the graph $G_{adg}$, the architecture model $G_{hma}$ and it outputs a list schedule $\sigma_a$ for $G_{adg}$ under resource and communication constraints. The list-scheduling algorithm consists of two phases:

– **Prioritizing Phase.** This phase requires the scheduling rank (i.e. the priority) of each node of $G_{adg}$ to be calculated firstly. The scheduling rank of a node $i_k \in V'$ is calculated by a recursive function *rank* given by:

$$rank(i_k) = \max_{j_{k'} \in pred(i_k)} \left\{ rank(j_{k'}) + \delta_{j_{k'}} \right\} \tag{5.13}$$

where $pred(i_k)$ is the set of immediate predecessors of $i_k$. For every node without predecessors, $rank(i_k)$ is set to zero. Secondly, a scheduling list is generated by sorting the nodes of $G_{adg}$ by increasing order of scheduling ranks. Tie-breaking is randomly performed to sort the nodes with equal ranks. It can easily be shown that the increasing order of scheduling ranks provides a topological order of the nodes, which is a linear order that preserves the dependency relations.

– **Mapping Phase.** Nodes are selected from the scheduling list by increasing order of the ranks, and each node is allocated to the available processing resource $x \in R$ that minimizes its earliest finishing time (EFT). To map a selected node on a selected processing resource, we use an insertion-based scheduling policy that tries to insert if possible the node in an earliest idle time slot of the resource (i.e an idle time interval between two already scheduled nodes on this resource) while ensuring the preservation of precedence constraints. Let $EFT(x, i_k)$ be the earliest finishing time of the node $i_k \in V'$ on the resource $x \in R$. This function is defined as follows:

$$EFT(x, i_k) = \max\left\{ avail(x), ready(x, i_k) \right\} + \Delta_{xi} \tag{5.14}$$

where $avail(x)$ is the earliest time at which the PU $x$ is available to execute a new node and $ready(x, i_k)$ is the time instant at which the node $i_k$ can be processed on the resource $x$. In order to fulfill precedence and communication constraints, we define $ready(x, i_k)$ by the following equation:

$$ready(x, i_k) = \max_{j_{k'} \in pred(i_k)} \left\{ AFT(j_{k'}) + \Gamma_{proc(j_{k'})x} \right\} \tag{5.15}$$

where $pred(i_k)$ is the set of direct predecessors of $i_k$ in the graph $G_{adg}$, $AFT(j_{k'})$ is the actual finishing time of the node $j_{k'}$ and $proc(j_{k'})$ is the processing resource on which the node $j_{k'}$ is mapped to. For nodes without predecessors, $ready(x, i_k)$ is set to zero.

- **Step 4.** A valid SWP schedule $\sigma$ of period $\lambda$ for $G_{hsdf}$ is derived under the resource and communication constraints of $G_{hma}$. In order to derive this schedule, we first calculate the period $\lambda$ with the information provided by the schedule $\sigma_a$ and the communication matrix $\Gamma$, where $\lambda$ is the minimum time required to process both the computations and communications of every actor in a single iteration of $G_{hsdf}$. Using this period we derive the schedule $\sigma$ with the following cyclicity equation:

$$\sigma(n, i_k) = \sigma_a(i_k) + n \cdot \lambda. \tag{5.16}$$

## Correctness of HCS

**Theorem 5.1** (correctness of HCS)**.** *The schedule $\sigma$ of period $\lambda$ obtained with Algorithm 3 satisfies both resources, cyclicity, precedence and communication constraints.*

*Proof.* Resource constraints are obviously met because of the list scheduling algorithm (i.e. Algorithm 4), which ensures that any node in $G_{adg}$ is assigned to a single resource and the execution of firings allocated to the same resource are not overlapped. The respect of cyclicity constraints is ensured by equation 5.16, which guarantees that the computations and communications of every node $i_k \in V'$ are processed cyclically according to the period $\lambda$. Now to ensure that precedence and communication constraints are fulfilled, let us consider a channel $e' = (i_k, j_{k'}) \in E'$ and let us assume that the firings of nodes $i_k$ and $j_{k'}$ are respectively assigned to resources $x^*$ and $y^*$.

On one hand, if $e'$ is a direct dependency channel (i.e. $e' \in E_{adg}$), then:

$$\sigma(n, j_{k'}) \geq \sigma(n, i_k) + \Delta_{x^* i} + \Gamma_{x^* y^*}.$$

By equation (5.16), the inequality above can be rewritten and simplified as:

$$\sigma_a(j_{k'}) \geq \sigma_a(i_k) + \Delta_{x^* i} + \Gamma_{x^* y^*}.$$

Since $\sigma_a$ is the schedule of the acyclic dependency graph $G_{adg}$, this inequality always hold and thus, precedence and communication constraints are satisfied.

On the other hand, if $e'$ is a loop-carried dependency channel (i.e. $e' \notin E_{adg}$), then:

$$\sigma(n + m_0(e'), j_{k'}) - \sigma(n, i_k) \geq \Delta_{x^*i} + \Gamma_{x^*y^*}$$

By equation (5.16), this inequality can be rewritten as:

$$\sigma_a(j_{k'}) + (n + m_0(e')) \cdot \lambda - \sigma_a(i_k) - n \cdot \lambda \geq \Delta_{x^*i} + \Gamma_{x^*y^*}.$$

Simplifying further and reordering, we get:

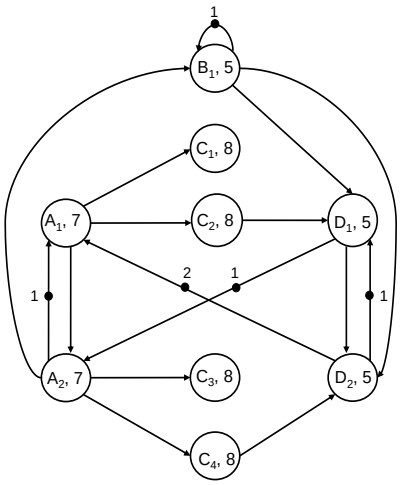
$$\sigma_a(j_{k'}) \geq \sigma_a(i_k) + \Delta_{x^*i} + \Gamma_{x^*y^*} - m_0(e') \cdot \lambda.$$

This implies that, the schedule $\sigma$ fulfils the precedence and communication constraints induced by loop-carried dependency channels of $G_{hsdf}$ and the proof is achieved. ■

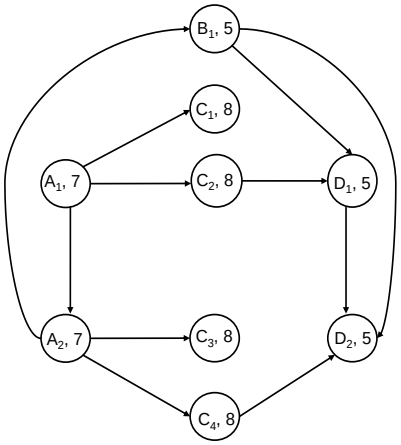
## Illustration of HCS

In order to illustrate the different steps of the heuristic HCS, let $G_{sdf}$, $G_{hsdf}$ and $G_{hma}$ respectively be the non-timed SDF graph shown in figure 2.6b, the equivalent HSDF graph of this graph (refer to figure 3.2) and the multiprocessor architecture of our running example (refer to figure 3.8).

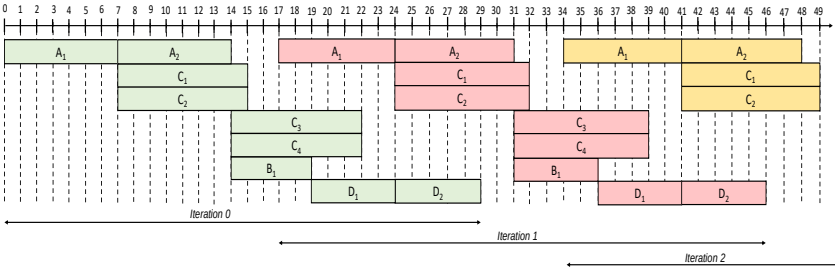
- **Step 1:** $G_{hsdf}$ is set into a timed HSDF graph $G^t_{hsdf}$. Figure 5.3a depicts the graph $G^t_{hsdf}$, where for any value of $k$, the time budgets of nodes are given by $\delta_{A_k}$=7, $\delta_{B_k}$=5, $\delta_{C_k}$=8, and $\delta_{D_k}$=5. Considering these time budgets, an optimal SWP schedule $\sigma_\infty$ of period $\lambda\infty$ is calculated for $G^t_{hsdf}$ without considering resource and communication constraints. Figure 5.3c presents the schedule $\sigma_\infty$ of period $\lambda_\infty = 17$ for our running example. This schedule is obtained by instantiating and solving the linear programming model ($P_2$).
- **Step 2:** an acyclic dependency graph $G_{adg}$ is generated by deleting all the loop-carried dependency channels in $G^t_{hsdf}$. Figure 5.3b illustrates this acyclic dependency graph for our running example.



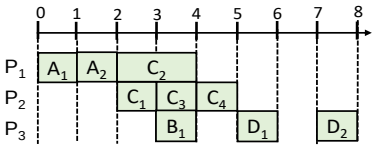

(a) Timed homogeneous graph $G^t_{hsdg}$



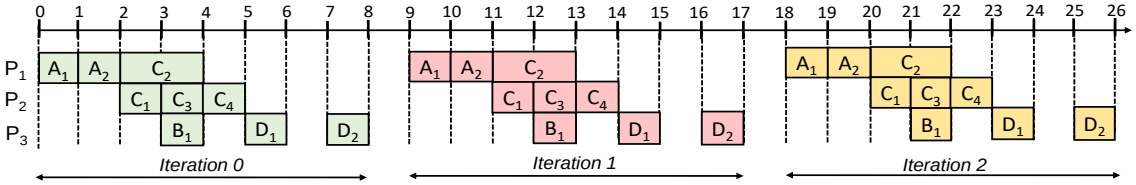

(b) Acyclic dependency graph $G_{adg}$




(c) An optimal SWP schedule $\sigma_\infty$ of $G^t_{hsdf}$




(d) List schedule of $G_{adg}$ under resource and communication constraints




(e) A SWP $\sigma$ of $G_{hsdf}$ under resource and communication constraints

Figure 5.3: Illustration of the Heuristic HCS.

- **Step 3:** Algorithm 4 is used to perform the list schedule of $G_{adg}$ under the resource and communication constraints of $G_{hma}$. The first phase (i.e. the prioritizing phase) of this algorithm consists in generating a scheduling list where the nodes of $G_{adg}$ are sorted by increasing order of rank. Table 5.1 shows an example of scheduling list for our running example. The second phase of Algorithm 4 (i.e. the mapping phase) consists to schedule and to map each node of the scheduling list on the processing unit $G_{hma}$ that minimizes the earliest finishing time. Figure 5.3d depicts the schedule $\sigma_a$ obtained with Algorithm 4 for the graph $G_{adg}$ of our running example.

Table 5.1: Scheduling list for the acyclic dependency graph of Fig. 5.3b

| $i_k$ | $A_1$ | $A_2$ | $C_1$ | $C_2$ | $B_1$ | $C_3$ | $C_4$ | $D_1$ | $D_2$ |
|---|---|---|---|---|---|---|---|---|---|
| $rank(i_k)$ | 0 | 7 | 7 | 7 | 14 | 14 | 14 | 19 | 24 |

- **Step 4:** Using the information provided by the list schedule $\sigma_a$, the communication matrix $\Gamma$, and the SWP schedule $\sigma_\infty$, we derive a valid SWP schedule $\sigma$ of period $\lambda$ for $G_{hsdf}$ under resource and communication constraints. Figure 5.3e depicts the schedule $\sigma$ of period $\lambda = 9$ returned by the heuristic HCS for the application graph and the architecture of our running example. It can easily be proved that this schedule satisfies both cyclicity, resource, precedence and communication constraints and a new iteration of $G_{hsdf}$ occurs according to the period $\lambda = 9$.

In order to validate the heuristic HCS, we need to study and to characterize the performance achievable by the heuristic HCS according to the performance achievable by an ILP solver for different instances of SDF graphs and multiprocessor architectures. In the next chapter, we will present and discuss in detail the performance results obtained.

## 5.4 Conclusion

In this chapter, we have presented an ILP model and a heuristic denoted by HCS for the scheduling and throughput optimization problem of SDF graphs on heterogeneous multiprocessor architectures under resource and communication constraints. The next chapter will be dedicated to the performance evaluation and discussion.

# CHAPTER 6

# Validation

## Contents

## 6.1 Introduction

In this chapter, we present the performance results for the heuristic HCS. Performance evaluation have been achieved using synthetic benchmarks and real-world application benchmarks. All experiments were performed on a PC Intel(R) core TM i7-7600U running at 2.80GHz with 16GB of RAM. In order to calculate an exact scheduling solution for a given benchmark, we use the ILP solver of CPLEX 12.5.0 and OPL script language to instantiate and solve our ILP formulation for the benchmark.

## 6.2 Evaluation Metrics

Our experiments are based on the following performance metrics:

- **Solving Time:** the time to find a scheduling solution.
- **Bound Gap (BG):** this metric is the average ratio between the scheduling solutions obtained with HCS and the ILP solver of CPLEX. It enables to evaluate how far the throughput of schedules returned by the heuristic HCS is from the throughput of schedules generated with the ILP solver. The value of BG is given by:

$$BG = \frac{\lambda_{hcs} - \lambda_{cplex}}{\lambda_{cplex}} \times 100 \tag{6.1}$$

  where $\lambda_{hcs}$ and $\lambda_{cplex}$ are respectively the periods of scheduling solutions obtained with HCS and CPLEX. A low percentage of BG means that the scheduling solution obtained with HCS is very close to the solution obtained with the ILP solver. Conversely, a high percentage of BG implies that the solution obtained with HCS is suboptimal compared to that obtained with the ILP solver.

- **Speedup.** Speedup is defined as the sequential execution time of a SDFG divided by the latency ($\mathcal{T}$) of this graph, where $\mathcal{T}$ is the amount of time required to execute all the firings of every actor in each stable iteration of the graph. To calculate the sequential execution time of a SDFG, we assign the firings of every actor to the single PU that minimizes the cumulative computation costs and we characterize the speedup by the following equation:

$$Speedup = \frac{\min\limits_{x \in R} \left[ \sum\limits_{i \in V} q_i \times \Delta_{xi} \right]}{\mathcal{T}} \tag{6.2}$$

Table 6.1: Results of Average Solving Times (sec) for different synthetic benchmarks

| Benchmarks | NP=2 | | NP=4 | | NP=8 | | NP=16 | |
|---|---|---|---|---|---|---|---|---|
| – | HCS | ILP | HCS | ILP | HCS | ILP | HCS | ILP |
| NA=10 | 2.44 | 176.45 | 7.29 | 302.36 | 18.77 | 4287.25 | 119.58 | 12480.02 |
| NA=20 | 4.17 | 208.11 | 9.03 | 543.08 | 22.68 | 6745.97 | 172.71 | ∞ |
| NA=30 | 5.92 | 324.21 | 11.82 | 675.57 | 27.11 | 8123.66 | 189.6 | ∞ |
| NA=40 | 6.98 | 395.64 | 14.09 | 793.41 | 35.79 | 9745.71 | 201.2 | ∞ |
| NA=50 | 8.71 | 417.08 | 16.78 | 906.5 | 43.16 | 12456.73 | 224.38 | ∞ |
| NA=100 | 10.16 | 745.67 | 22.27 | 1203.66 | 53.19 | 16289.92 | 378.45 | ∞ |

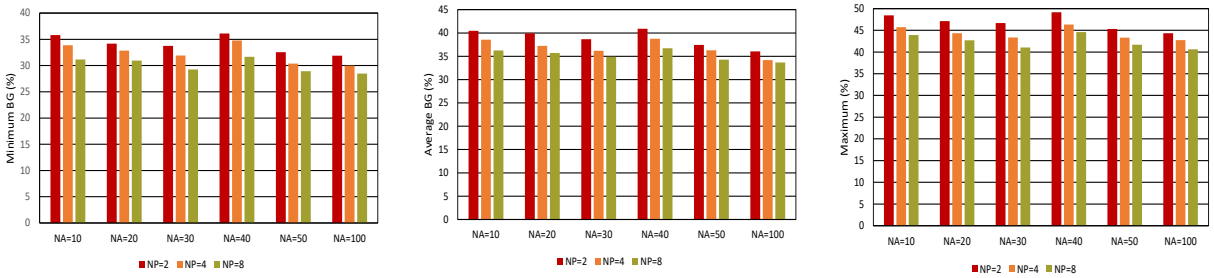
## 6.3 Experiments with Synthetic Benchmarks

### 6.3.1 Benchmarks Generation

In order to achieve a broad range of experiments, we have generated synthetic SDF graphs using Turbine[1], a multi-functional tool presented in [27], that randomly generates consistent, live, cyclic and acyclic dataflow graphs. General settings for a SDFG shape are the number of actors (NA) and the outgoing degree (outDeg) of an actor. In order to generate heterogeneous computing architecture for each SDF graph, we have adapted Turbine with a function that takes as inputs five parameters (NP, HFS, HFC, MCC, MIPCC) and outputs asymmetric computation and communication cost matrices as described in Figure 3.8b. The parameter NP stands for the number of processing units (PUs) on a given architecture. HFS and HFC stand respectively for the heterogeneity factor for PUs speed and the heterogeneity factor for inter-PUs communication. A high percentage of HFS implies high difference in computation costs for the PUs and a high percentage of HFC implies high difference in communication costs. MCC and MIPCC stand respectively for the mean computation cost of the input SDFG instance and the mean inter-PUs communication cost. In order to generate the computation cost matrix, the generation function selects randomly a mean computation cost $\bar{\Delta}_i$ of every actor $i$ from an uniform distribution in the range of 0 to $0.2 \times MCC$ and then, the computation cost of every actor on every PU is randomly selected from an uniform distribution of range $\bar{\Delta}_i \times \left(1 - \frac{HFS}{2}\right) \leq \Delta_{xi} \leq \bar{\Delta}_i \times \left(1 + \frac{HFS}{2}\right)$. Replacing $MCC$, $HFS$, $\Delta_{xi}$, $\bar{\Delta}_i$ respectively by MIPC, HFC, $\Gamma_{xy}$, $\bar{\Gamma}_{xy}$ in this distribution, we generate the communication cost matrix. By definition, we set $\Gamma_{xy} = \Gamma_{yx}$ for each pair $(x, y)$ of PUs and whether $x$ is equal to $y$, we set the value of $\Gamma_{xy}$ to 0.

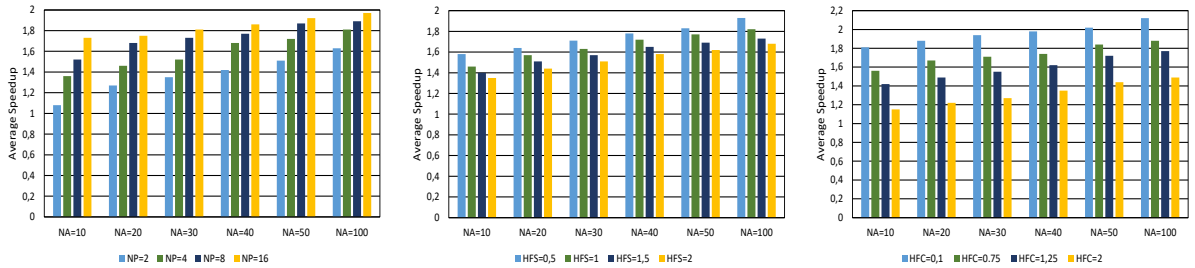
The following sets were considered for the experiments: NA={10, 20, 30, 40, 50,

[1] https://github.com/bbodin/turbine




(a) Minimum values of BG (b) Average values of BG (c) Maximum values of BG

Figure 6.1: Results of BG for synthetic Benchmarks




(a) Speedup for different values of NP (b) Speedup for different values of HFS (c) Speedup for different values of HFC

Figure 6.2: Results of average Speedup for synthetic Benchmarks

100}, outDeg={1,2}, NP={2, 4, 8, 16}, HFS={0.5, 1, 1.5, 2}, HFC={0.1, 0.75, 1.25, 2}, MCC={15, 30, 60} and MIPCC={10, 25, 40}. The combination of these sets gives 6912 different benchmarks. The SDF graph of each benchmark is transformed into an equivalent HSDF graph using Algorithm 1.

### 6.3.2 Performance Results

**Time complexity.** In order to characterize the time complexity of the heuristic HCS, we compared the solving times of the ILP solver with the solving times of the heuristic. We have limited the running time of CPLEX to 8 hours. Table. 6.1 plots the results of the average solving times for our benchmarks. The ILP solver was able to find a scheduling solution for all multiprocessor architectures except for the 16-PUs architectures, where $\infty$ means that the solver fails to find a scheduling solution within 8 hours. Conversely, the scalability of HCS is easily visible and in some cases the heuristic is approximately $300 \times$ *faster* than the ILP solver.

**Throughput achievement.** In order to evaluate how far the throughput measured with the heuristic HCS is from the throughput measured with the ILP, we studied the variations

Table 6.2: Benchmarks Characteristics

| Benchmarks | Description | Number of Actors | Stateful actors |
|---|---|---|---|
| bitonicSort | Recursive implementation of the bitonic sorting network | 61 | 7 |
| fft | Fast Fourier Transform kernel | 17 | 2 |
| filterBank | A filter bank to perform multi-rate signal processing | 53 | 6 |
| radar | Radar Array Front-End | 54 | 6 |
| tde | Time delay equalization | 42 | 5 |

of BG for the synthetic benchmarks according to different values of NA and NP. Figure 6.1 plots the results of minimum, average and maximum values of BG. For all the benchmarks, it can be observed that the average values of BG decrease as the values of NP increase. The interpretation of this result is that the throughput measured with heuristic HCS are getting closer to the throughput measured with the ILP solver as the number of PUs is getting greater. Moreover, it can globally be observed that the minimum and maximum values of BG respectively vary approximately in the ranges of 28 —36% and 40—48% as the value of NP increases. This means that the throughput of scheduling solutions returned by HCS are getting closer to the throughput of scheduling solutions obtained with the ILP solver in the ranges of 64 —72% in the best case and 52—60% in the worst case.

**Speedup and parallelism exploitation.** Now, if we want to characterize the relative performance of HCS with respect to hardware features, we should study the variations of speedup for different types of heterogeneous multiprocessor architectures. For this purpose, we first set the parameters HFS and HFC respectively to 1.5 and 1.25 to evaluate the average speedup of HCS for different values of NP and then, we set the parameter NP to 4 to evaluate the speedup of HCS with respect to different values of HFS and HFC. Figure 6.2 shows the results obtained. In figure 6.2a, it can be observed that the average speedup of all the benchmarks increases as when as the values of NP get increased. Conversely, in figure 6.2b and figure 6.2c we respectively observed that, the speedup of the heuristic gradually decreases when the values of HFS and HFC are increased but it is still greater to 1. This means that the speedup of the heuristic get improved when the number of processing resources get greater. However, whether these resources have a high variability in computation and communication costs of actors, the speedup of the heuristic may decrease but will still be greater to 1. The interpretation of these results is that, if we take a higher risk to increase the value of HFS and HFC, we will certainly loose task, data or pipeline parallelism but, there is still a guarantee that the latency of scheduling solutions returned by the heuristic could not be worse than the sequential execution time of the application graphs.

Table 6.3: Results of Average Solving Times (sec) of HCS versus ILP solver

| Benchmarks | NP=2 | | NP=4 | | NP=8 | | NP=16 | |
|---|---|---|---|---|---|---|---|---|
| – | HCS | ILP | HCS | ILP | HCS | ILP | HCS | ILP |
| bitonicSort | 5.81 | 776.72 | 15.97 | 3391.98 | 34.13 | 9017.86 | 114.05 | ∞ |
| fft | 3.44 | 323.29 | 10.98 | 1798.13 | 21.13 | 6123.16 | 64.61 | 18765.57 |
| filterBank | 5.62 | 747.08 | 15.18 | 3065.29 | 32.66 | 8656.55 | 99.08 | ∞ |
| radar | 5.76 | 756.65 | 15.11 | 3168.76 | 33.25 | 8659.23 | 102.23 | ∞ |
| tde | 5.12 | 685.64 | 14.01 | 2891.86 | 28.75 | 7745.71 | 82.24 | ∞ |

## 6.4 Experiments with StreamIt Benchmarks

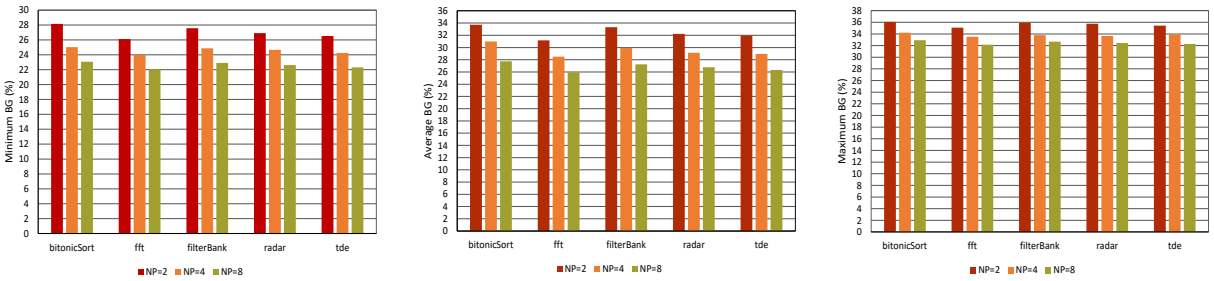
In addition to experiments performed with the synthetic benchmarks, we have also performed experiments with real-world applications to validate the performance of HCS.

### 6.4.1 StreamIt Benchmarks

Experiments were performed with the application benchmarks of StreamIt [43]. These benchmarks are streaming applications that embed the common properties (data, task and pipeline parallelisms) of loop-intensive applications. Table 6.2 gives a brief description of the chosen benchmarks. A detailed description of each of these benchmarks is given in [43]. We set the number of stateful actors for each application benchmark to approximately 10% of the total number of actors. In order to generate asymmetric computation and communication costs for the actors of each benchmark, we have adapted the StreamIt compiler with our architecture generation function previously described in section 5.3.1 and we consider the same values for the parameters of this function. For each benchmark and each architecture configuration, we ran both the heuristic and the ILP solver. In this experiment, we have also limited the running time of the ILP solver to 8 hours.

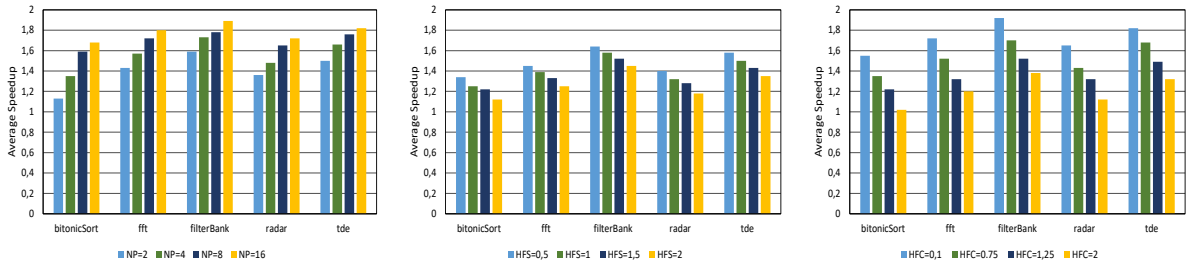
### 6.4.2 Performance Results

**Time complexity.** Table. 6.3 plots the results of average solving times of the heuristic HCS and the ILP solver of CPLEX. On one hand, when the number of processing resources (i.e. NP) is lower or equal to 8 the ILP solver is able to find a scheduling solution for all the streamIt benchmarks. However, on the other hand, when the number of processor is equal to 16, the ILP fails to find a scheduling solution for all the benchmark except for "fft ". Conversely, the scalability of the heuristic HCS find a solution for any number of processor and in some cases the heuristic is approximately $355\times faster$ than the ILP solver.




(a) Minimum values of BG (b) Average values of BG (c) Maximum values of BG

Figure 6.3: Results of BG for StreamIt Benchmarks




(a) Speedup for different values of NP (b) Speedup for different values of HFS (c) Speedup for different values of HFC

Figure 6.4: Results of average speedup for StreamIt Benchmarks

**Throughput achievement.** In order to evaluate the throughput gap between the scheduling solutions returned by the ILP solver and the heuristic HCS, we plot the results of BG for the StreamIt benchmarks according to differents values of NP. Figure 6.3 depicts these results. We have observed similar results like those observed for the synthetic benchmarks. Actually, for all the streamIt benchmarks, the values of BG decrease as the number of processing resources increases and the minimum and maximum values of BG respectively range approximately in 22—28% and 31—36%. The worst gap has been observed with "bitonicSort" where the average value of BG decreases slowly from approximately 33% to 27% as when as the number of resource get increased.

**Speedup and parallelism exploitation.** In order to measure the performance of the heuristic HCS with respect to hardware features, we measured the speedup of HCS for the streamIt benchmarks according to different types of multiprocessor architectures. In these experiments, we first considered architectures generated with HFS=1.5 and HFC=1.25 to plot the speedup of the heuristic ans then, we considered the 4-processors architectures —i.e. architectures where NP=4 —to plot the speedup of the heuristic for different values of HFS and HCS. Figure 6.4a, figure 6.4c and figure 6.4b respectively show the results of average speedup for the heuristic according to different values of NP, HFS and HFC. On

the one hand, the speedup increases in the range of 1.1 —1.9 as the number of processing resources (i.e. NP) increases. On the other hand, the speedup decreases in the range of 1.1 —1.65 and respectively 1.1 —1.9 respectively as the values of HFS and HFC increases. These results are similar to the those observed with the synthetic application graphs and they validate the performance of the heuristic.

## 6.5 Conclusion

In this chapter, we presented the performance of the ILP formulation and the heuristic HCS for the software pipelined schedule of SDF graphs on heterogeneous multiprocessor architectures under resource and communication constraints. Experiments were performed both with synthetic application graphs and application benchmarks of StreamIt. Experimental results show that the heuristic performs better for multiprocessor architectures with large number of processing resources and more is the number of processing resources on a given architecture, better is the performance of the heuristic. In terms of throughput achievement, the heuristic was able to generate scheduling solutions with an average gap ranging from 25% to 42%, which means that in the average case, the scheduling solutions obtained with the ILP solver of CPLEX is close to the optimal solutions in the range of 68% to 75%. For all the benchmarks and multiprocessor architectures considered, the speedup of heuristic vary in the range of 1 to 1.9. Although the heterogeneity degree of processing resources may have an effect of decreasing the speedup of the heuristic, there is a guarantee that the latency of scheduling solutions returned by the heuristic could not be worse than the sequential execution time of application graphs.

# CHAPTER 7

# Conclusion & Open Challenges

## Contents

## 7.1 Conclusion

Cyber-physical systems (CPSs) are increasingly implemented in several application fields to address many technical challenges. These systems are composed by a set of loop-intensive applications and heterogeneous multiprocessor architectures with a fixed number of processing resources, which are connected through different networks media.

A key activity in the design stage of CPSs is the deployment of loop-intensive applications on the heterogeneous multiprocessor architectures. The goal of this design activity is to predict the timing behaviour of applications and to provide performance guarantees at design stage. For this purpose, formal models of computation that deal with time, concurrency and parallelism are required to generate static order schedules for the applications while ensuring highest performance under the resource and communication constraints of architectures. In order to achieve this need, we have used synchronous dataflow (SDF) model of computation in this thesis and we have design exact and approximated software pipelined (SWP) scheduling techniques to execute these models under the resources and communication constraints of heterogeneous multiprocessor architectures. Our scheduling techniques are based on a set of provable mathematical theories and they exploit efficiently the parallelism embedded in the SDF models while providing performance guarantees in terms of throughput and latency.

In chapter 4, we have established a set of lemmas and theorems to characterize the admissible SWP schedules for timed SDF graphs. Using the established theorems, we have proposed linear programming models to find the admissible SWP schedules that achieve optimal throughput/latency for timed SDF graphs. In chapter 5, we have shown that the SWP schedules that achieve optimal throughput for a SDF graph under the resource and communication constraints of a heterogeneous multiprocessor architecture can be characterized and calculated with an integer linear programming (ILP) model. The proposed ILP model explore different levels of parallelism (task, data and pipeline) in SDF graphs while scheduling these graphs. In this chapter, we have also proposed a decomposed software pipelining heuristic that generates approximated SWP scheduling solutions for SDF graphs under resource and communication constraints. Thanks to experiments performed with synthetic and application benchmarks, our ILP model and heuristic are validated in chapter 6. To the best of our knowledge this thesis is the first that tackles the SWP scheduling of SDF graphs under the resources and communication constraints of heterogeneous multiprocessor architectures.

## 7.2 Open Challenges

The theoretical foundations put forward in this thesis provides a basis for future research directions. In this section, we present three interesting challenges that could be investigated to improve and extend the works achieved in this thesis.

### 7.2.1 List scheduling heuristics for throughput improvement

Thanks to the experimental results presented in chapter 6, we have shown that the scheduling solutions obtained with the heuristic HCS achieve a throughput gap bounded at 50% to the throughput achievable by the scheduling solutions obtained with our ILP formulation. In order to reduce this throughput gap, we need to investigate new list-scheduling algorithms that can improve the length of the schedule generated by algorithm 4. Since the period (implicitly the throughput) of SWP scheduling solutions obtained with the heuristic HCS depends on the length of schedules returned by algorithm 4, an improvement of scheduling solutions obtained with this algorithm will obviously improve the throughput of SWP scheduling solutions obtained with the heuristic HCS.

### 7.2.2 Scheduling under storage capacity

Apart from resource and communication constraints, the storage capacity of communication links in CPS architectures can be bounded. Therefore, the scheduling strategy developed in this thesis has to assign not only actors to processing units but also FIFO channels to the communication links while ensuring that the storage capacity is not overflowed. This new constraint can reduce the searching-space for scheduling solutions that achieve optimal throughput/latency and increase the time complexity to find a solution. In order to overcome this problem, future research works should investigate time-efficient heuristics that deal both with storage, resource and communication constraints while providing good performance guarantees.

### 7.2.3 Real-time scheduling

The major drawback of static schedules (and SWP schedules in particular) is their inflexibility and difficult maintainability. Hence, to execute a SDF graph on a heterogeneous architecture, real-time scheduling policies should be investigated. Unlike in SWP schedules, each actor in a real-time schedule is mapped to a periodic real-time task; therefore, the firing of a given actor is strictly periodic (as opposite to SWP schedule). One advan-

tage of this scheduling approach is that there exist a set of provable mathematical theories (such as rate-monotonic, fixed-priority scheduling, earliest-deadline first, etc) that can be used to decide whether or not a dataflow specification can be scheduled on a given architecture. Using real-time scheduling policies to implement dataflow graphs under resource and communication constraints has been subject of only few works. Investigating this research direction can be very helpful to improve the works achieved in this thesis.

# Personal Bibliography


- **P. Glanon**, S. Azaiez, C. Mraidha, “HCS: A Cyclic Scheduling Heuristic for Deploying Loop-Intensive Applications on Heterogeneous Multiprocessor Architectures”, International Journal of Systems Architectures (JSA), [under revision process]
- **P. Glanon**, S. Azaiez, C. Mraidha, “Cyclic Scheduling of Loop-Intensive Applications on Heterogeneous Multiprocessor Architectures,” RTCSA’ 2020: the 26th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications. [Accepted]
- **P. Glanon**, S. Azaiez and C. Mraidha, “Estimating Latency for Synchronous Dataflow Graphs Using Periodic Schedules,” in proceedings of VECoS 2019: the 13th International Conference on Verification and Evaluation of Computer and Communication Systems, pp 79-94.
- **P. Glanon**, S. Azaiez and C. Mraidha , “Analyzing throughput for Cyber-Physical Production System modeled with synchronous Dataflow ,” Proceedings of the Cyber-Physical Systems PhD Workshop 2019, an event held within the CPS Summer School ”Designing Cyber-Physical Systems - From concepts to implementation”, Alghero, Italy, September 23, 2019. CEUR Workshop Proceedings 2457, CEUR-WS.org 2019.
- **P. Glanon**, S. Azaiez and C. Mraidha, ”A modular interoperability layer for connecting the business and manufacturing systems,” 2018 14th IEEE International Workshop on Factory Communication Systems (WFCS), Imperia, 2018, pp. 1-4.

# Bibliography


[1] J. A. Stankovic, Misconceptions about Real-Time Computing: A Serious Problem for Next Generation Systems, IEEE Computer, 21(10), pp. 10-19, October 1988.

[2] J. Herwan, S. Kano, R. Oleg, H. Sawada and N. Kasashima, "Cyber-physical system architecture for machining production line," 2018 IEEE Industrial Cyber-Physical Systems(ICPS), St. Petersburg, 2018, pp. 387-391.

[3] C. Shih, J. Chou, N. Reijers and T. Kuo, "Designing CPS/IoT applications for smart buildings and cities," in IET Cyber-Physical Systems: Theory & Applications, vol. 1, no. 1, pp. 3-12.

[4] J. Lee, B. Bagheri, and H.-A. Kao, "A cyber-physical systems architecture for industry 4.0-based manufacturing systems," Manufacturing Letters, vol. 3, pp. 18–23, 2015.

[5] N. Jazdi, "Cyber physical systems in the context of Industry 4.0," 2014 IEEE International Conference on Automation, Quality and Testing, Robotics, Cluj-Napoca, 2014, pp. 1-4.

[6] L. Monostori, "Cyber-physical production systems: Roots, expectations and R&D challenges," Procedia CIRP 17, pp. 9-13, 2014

[7] Haque, S. A., Aziz, S. M., & Rahman, M. (2014). Review of Cyber-Physical System in Healthcare. International Journal of Distributed Sensor Networks.

[8] Edward A. Lee and Sanjit A. Seshia, Introduction to Embedded Systems, A Cyber-Physical Systems Approach, Second Edition, MIT Press, ISBN 978-0-262-53381-2, 2017.

[9] Lee, Edward, "Cyber Physical Systems: Design Challenges", University of California, Berkeley Technical Report No. UCB/EECS-2008-8.

[10] Sanjay Raina. Virtual Shared Memory: A Survey of Techniques and Systems. Dec. 1992.

[11] Michael J Flynn. Some Computer Organizations and Their Effectiveness. IEEE Transactions on Computers, C-21 :948–960, 1972.

[12] John Von Neumann. First Draft of a Report on the EDVAC. Technical Report 1, 1945.

[13] G.M. Amdahl. Validity of the single processor approach to achieve large scale computing capabilities. In American Federation of Information Processing Societies Conference, AFIPS 67, Proceedings, pages 483–485. Thomson Book Company, 1967.

[14] M. D. Hill and M. R. Marty, "Amdahl's law in the multicore era," Computer, vol. 41, no. 7, pp. 33–38, Jul. 2008. [Online]. Available: http://dx.doi.org/10.1109/MC.2008.209

[15] J. Diaz, C. Munoz-Caro, and A. Nino, "A survey of parallel programming models and tools in the multi and many-core era," Parallel and Distributed Systems, IEEE Transactions on, vol. 23, no. 8, pp. 1369–1386, Aug 2012.

[16] Edward A. Lee. The Problem with Threads. Computer, 39(5) :33–42, May 2006.

[17] OpenMP Application Program Interface Version 3.0. Technical report, 2008.

[18] Message Passing Interface Forum. MPI : A Message-Passing Interface Standard. International Journal of Supercomputer Applications, 8 :623, 1994.

[19] Sur, Sayantan; Koop, Matthew J.; Panda, Dhabaleswar K. (4 August 2017). "MPI and communication-High-performance and scalable MPI over Infini Band with reduced memory usage". High-performance and Scalable MPI over InfiniBand with Reduced Memory Usage: An In-depth Performance Analysis. ACM. p. 105.

[20] David R. Butenhof. Programming With Posix Threads. 1997.

[21] K. Group. (2013) Opencl-the open standard for parallel programming of heterogeneous systems. [Online]. Available: http://www.khronos.org/opencl/

[22] Ait El Cadi, A. Souissi, O. Ben Atitallah, R. et al. Mathematical programming models for scheduling in a CPU/FPGA architecture with heterogeneous communication delays. Journal of Intelligent Manufacturing (2018) 29: 629.

[23] Jad Khatib, Alix Munier-Kordon, Enagnon Cedric Klikpo, and Kods Trabelsi-Colibet. 2016. Computing latency of a real-time system modeled by Synchronous Dataflow

Graph. In Proceedings of the 24th International Conference on Real-Time Networks and Systems (RTNS '16). ACM, New York, NY, USA, 87-96.

[24] Youen Lesparre. Efficient evaluation of mappings of dataflow applications onto distributed memory architectures. Mobile Computing. Université Pierre et Marie Curie - Paris VI, 2017. English. NNT: 2017PA066086. tel-01624553

[25] M. Avinash and D. Gregg. "Heuristics on Reachability Trees for Bicriteria Scheduling of Stream Graphs on Heterogeneous Multiprocessor Architectures." ACM Transactions on Embedded Computing Systems, Vol. 14, No. 2, Article 23, Publication date: February 2015.

[26] T. Schwarzer, J. Falk, M. Glass , J. Teich, C. Zebelein, and C. Haubelt. Throughput-optimizing Compilation of Dataflow Applications for MultiCores using Quasi-Static Scheduling. In S. Stuijk, editor, Proc. of the 18th International Workshop on Software and Compilers for Embedded Systems, SCOPES'15, pages 68-75, Berlin, Germany, June 2015. ACM.

[27] Bodin, B., Lesparre, Y., Delosme, J.-M., and Munier-Kordon, A. (2014). Fast and efficient dataflow graph generation. In Proceedings of the 17th International Workshop on Software and Compilers for Embedded Systems, pages 40-49. ACM.

[28] R. de Groote, J. Kuper, H. Broersma and G. J. M. Smit, "Max-Plus Algebraic Throughput Analysis of Synchronous Dataflow Graphs," 2012 38th Euromicro Conference on Software Engineering and Advanced Applications, Cesme, Izmir, 2012, pp. 29-38.

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

[30] Bruno Bodin, Alix Munier-Kordon, et Benoît Dupont de Dinechin. K-Periodic Schedules for Evaluating the Maximum Throughput of a Synchronous Dataflow Graph. In International Conference on Embedded Computer Systems : Architectures, Modeling, and Simulation, SAMOS XII, pages 152–159, 2012.

[31] Benabid-Najjar, A., Hanen, C., Marchetti, O., and Munier-Kordon, A. (2012). Periodic schedules for bounded timed weighted event graphs. IEEE Transactions on Automatic Control, 57(5):1222–1232.

[32] Paul M. Carpenter, Alex Ramirez, and Eduard Ayguade. 2009. Mapping stream programs onto heterogeneous multiprocessor systems. In Proceedings of the 2009 international conference on Compilers, architecture, and synthesis for embedded systems (CASES '09). ACM, New York, NY, USA, 57-66.

[33] A. Udupa, R. Govindarajan, and M. J. Thazhuthaveetil. Software Pipelined Execution of Stream Programs on GPUs. In CGO, pages 200-209, 2009

[34] Sriram S, Bhattacharyya SS (2009) Embedded multiprocessors: scheduling and synchronization, 2nd edn. CRC Press, Boca Raton. doi:10.1201/9781420048025

[35] O. Marchetti et A. Munier Kordon A sufficient condition for the liveness of weighted event graphs European Journal of Operational Research,197(2), pp. 532-540, Sept 2009.

[36] O. Marchetti. and A. Munier-Kordon. Cyclic scheduling for the synthesis of embedded systems. In Y. Vivien and R. Frederic, editors, Introduction to scheduling, chapter 6. Chapman and Hall/CRC Press, 2009, pages 135–164.

[37] Claire Hanen. Cyclic scheduling. In Y. Vivien and R. Frederic, editors, Introduction to scheduling, chapter 5. Chapman and Hall/CRC Press, 2009, pages 109–131.

[38] Stuijk, S.: Predictable mapping of streaming applications on multiprocessors. Ph.D. thesis, Eindhoven University of Technology (2007).

[39] A. H. Ghamarian, M. C. W. Geilen, T. Basten, B. D. Theelen, M. R. Mousavi and S. Stuijk. Liveness and Boundedness of Synchronous Data Flow Graphs, 2006 Formal Methods in Computer Aided Design, San Jose, CA, 2006, pp. 68-75.

[40] M. I. Gordon, W. Thies, and S. Amarasinghe. 2006. Exploiting coarse-grained task, data, and pipeline parallelism in stream programs. SIGOPS Oper. Syst. Rev. 40, 151-162.

[41] E. H. M. Sha, "Parallel embedded systems: optimizations and challenges," Conference, Emerging Information Technology 2005., Taipei, 2005, pp. 4 pp.-, doi: 10.1109/EITC.2005.1544328.

[42] Sandnes, Frode & Sinnen, Oliver. (2005). A new strategy for multiprocessor scheduling of cyclic task graphs. IJHPCN. 3. 62-71.10.1504/IJHPCN.2005.007868.

[43] W. Thies, M. Karczmarek, and S. P. Amarasinghe. 2002. StreamIt: A language for streaming applications. In Proceedings of the 11th International Conference on Compiler Construction (CC'02). Springer, London, UK, 179-196.

[44] A. Dasdan, S. Irani, and R. K. Gupta. Efficient algorithms for optimum cycle mean and optimum cost to time ratio problems. In Design Automation Conference, pages 37–42, 1999.

[45] Y. Robert, A. Darte and P. Calland, "Circuit Retiming Applied to Decomposed Software Pipelining" in IEEE Transactions on Parallel & Distributed Systems, vol. 9, no. 01, pp. 24-35, 1998. doi: 10.1109/71.655240

[46] Bilsen, G., Engels, M., Lauwereins, R., and Peperstraete, J. A. (1995). Cyclo-static data flow. In 1995 International Conference on Acoustics, Speech, and Signal Processing. ICASSP-95., volume 5, pages 3255–3258. IEEE, 1995.

[47] Paul Feautrier. Fine-grain scheduling under resource constraints. In Languages and Compilers or Parallel Computing , number 892 in Lectures Notes in Computer Science, pages 1-15.Springer Verlag, 1994.

[48] F. Gasperoni and U. Schwielgelshohn, Generating Close to Optimum Loop-Schedules on Parallel Processors, Parallel Processing Letters, 4(4), 1994, pp. 391-403.

[49] R. Govindarajan, E. R. Altman, and G. R. Gao, "Minimizing Register Requirements Under Resource-constrained Rate-optimal Software Pipelining," in MICRO 27: Proc. of the 27th annual Intl. Symp. on Microarchitecture, 1994, pp. 85–94

[50] R. Govindarajan and G.R. Gao, "A Novel Framework for MultiRate Scheduling in DSP Applications",in Proceedings of the 1993 International Conference on Application Specific Array Processors, Venice, Italy, Oct. 25–27, 1993, pp. 77–88.

[51] R. Govindarajan and G.R. Gao. Rate Optimal Schedule for multi-rate DSP computation. ACAPS Technical Memo 61, School of Computer Science, McGill University, Montreal, Quebec, 1993.

[52] C.Hanen and A.Munier. Cyclic scheduling on parallel processors: an overview. Technical Report 822, Laboratoire de Recherche en Informatique, Universite de Paris Sud, Centre d'Orsay, 1993.

[53] J. Wang, Christine Eisenbeis. Decomposed software pipelining. [Research Report] RR-1838, INRIA.1993. ffinria-00074834f.

[54] Christine Eisenbeis, D. Windheiser. A New class of algorithms for software pipelining with resource constraints. [Research Report] RR-2033, INRIA. 1993. inria-00074638

[55] Qi Ning et Guang R. Gao. A novel framework of register allocation for software pipelining. Proceedings of the 20th ACM SIGPLAN-SIGACT symposium on Principles of programming languages - POPL '93, pages 29–42, 1993.

[56] Joseph T. Buck et Edward A. Lee. Scheduling dynamic dataflow graphs with bounded memory using the token flow model. In IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP-93), number September, pages 429–432,1993.

[57] Tsing-Fa Lee, Allen C-H. Wu, Daniel D. Gajski, and Youn-Long Lin. An effective methodology for functional pipelining. In Proceedings of the International Conference on Computer-Aided Design , pages 230-233, December 1992.

[58] Gilbert C. Sih et Edward A. Lee. A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures. IEEE Transactions on Parallel and Distributed Systems, 4(2) :175–187, 1993.

[59] M. R. Garey and D. S. Johnson, Computers and Intractability; A Guide to the Theory of NP-Completeness. New York, NY, USA: W. H. Freeman & Co., 1990

[60] Monica Lam. Software pipelining: An effective scheduling technique for VLIW machines. In Proc. of the SIGPLAN'88 Conf. on Programming Language Design and Implementation pages 318-328. Also in SIGPLAN Notices, 23(7), Jul. 1988.

[61] E. A. Lee and D. G. Messerschmitt, Synchronous data flow, Proceedings IEEE vol. 75, no. 9, pp. 1235-1245, Sept. 1987.

[62] E. A. Lee and D. G. Messerschmitt, "Static Scheduling of Synchronous Data Flow Programs for Digital Signal Processing," in IEEE Transactions on Computers, vol. C-36, no. 1, pp. 24-35, Jan. 1987.

[63] G. Kahn. The semantics of a simple language for parallel programming. In J.L. Rosenfeld, editor, Information Processing, pages 471-475. North Holland, Amsterdam, Aug 1974.

[64] F. Commoner, A.W. Holt, S. Even, and A. Pneuli. Marked directed graphs. Journal of Computer and System Sciences, 5(5):511–523, October 1971.

[65] R. Reiter. Scheduling parallel computations. Journal of the ACM, 15(4):590–599, Oct. 1968.

[66] Karp, R. M. and Miller, R. E. (1966). Properties of a model for parallel computations: Determinacy, termination, queueing. SIAM Journal on Applied Mathematics, 14(6):1390–1411.

[67] Bézout, Édition (1779). Théorie générale des équations algébriques. Paris, France: Ph.-D. Pierres.

[68] H. Wei, J. Yu, H. Yu, M. Qin and G. R. Gao, "Software Pipelining for Stream Programs on Resource Constrained Multicore Architectures," in IEEE Transactions on Parallel and Distributed Systems, vol. 23, no. 12, pp. 2338-2350, Dec. 2012, doi: 10.1109/TPDS.2012.41.

[69] S. Raskar, T. Applencourt, K. Kumaran and G. Gao, "Position Paper: Extending Codelet Model for Dataflow Software Pipelining using Software-Hardware Co-Design," 2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC), Milwaukee, WI, USA, 2019, pp. 640-645, doi: 10.1109/COMPSAC.2019.10280.

[70] A. Hatanaka, N. Bagherzadeh,A software pipelining algorithm of streaming applications with low buffer requirements,Scientia Iranica,Volume 19, Issue 3,2012,Pages 627-634,ISSN 1026-3098, https://doi.org/10.1016/j.scient.2011.08.034.

[71] Weiwen Jiang, Edwin H.-M. Sha, Xianzhang Chen, Lin Wu, Qingfeng Zhuge, Synthesizing distributed pipelining systems with timing constraints via optimal functional unit assignment and communication selection, Journal of Computational Science, Volume 26, 2018, Pages 332-343, ISSN 1877-7503, https://doi.org/10.1016/j.jocs.2017.03.020.

[72] Weiwen Jiang, Edwin H.-M. Sha, Xianzhang Chen, Lin Wu, Qingfeng Zhuge, Synthesizing distributed pipelining systems with timing constraints via optimal functional unit assignment and communication selection, Journal of Computational Science, Volume 26, 2018, Pages 332-343, ISSN 1877-7503, https://doi.org/10.1016/j.jocs.2017.03.020.

université PARIS-SACLAY

ÉCOLE DOCTORALE
Sciences et technologies de l'information et de la communication (STIC)

**Title:** Deployment of loop-intensive applications on heterogeneous multiprocessor architectures

**Keywords:** cyber-physical systems, multiprocessor scheduling, cyclic scheduling, static dataflow graphs, heterogeneous architectures, software pipelining, maximum throughput,

**Abstract:** Cyber-physical systems (CPSs) are distributed computing-intensive systems, that integrate a wide range of software applications and heterogeneous processing resources, each interacting with the other ones through different communication resources to process a large volume of data sensed from physical, chemical or biological processes. An essential issue in the design stage of these systems is to predict the timing behaviour of software applications and to provide performance guarantee to these applications. In order tackle this issue, efficient static scheduling strategies are required to deploy the computations of software applications on the processing architectures. These scheduling strategies should deal with several constraints, which include the loop-carried dependency constraints between the computational programs as well as the resource and communication constraints of processing architectures intended to execute these programs. Actually, loops being one of the most time-critical parts of many computing-intensive applications, the optimal timing behavior and performance of applications depends on the optimal schedule of loops structures enclosed in the computational programs. Therefore, to provide performance guarantee for the applications, the scheduling strategies should efficiently explore and exploit the parallelism embedded in the repetitive execution patterns of loops while ensuring the respect of resource and communications constraints of the processing architectures of CPSs. Scheduling a loop under resource and communication constraints is a complex problem. To solve it efficiently, heuristics are obviously necessary. However, to design efficient heuristics, it is important to characterize the set of optimal solutions for the scheduling problem. An optimal solution for a scheduling problem is a schedule that achieve an optimal performance goal. In this thesis, we tackle the study of resource-constrained and communication-constrained scheduling of loop-intensive applications on heterogeneous multiprocessor architectures with the goal of optimizing throughput performance for the applications. In order to characterize the set of optimal scheduling solutions and to design efficient scheduling heuristics, we use synchronous dataflow (SDF) model of computation to describe the loop structures specified in the computational programs of software applications and we design software pipelined scheduling strategies based on the structural and mathematical properties of the SDF model.

université PARIS-SACLAY

ÉCOLE DOCTORALE
Sciences et technologies de l'information et de la communication (STIC)

**Titre:** Déploiement d'applications à boucles intensives sur des architectures multiprocesseur hétérogènes

**Mots clés:** Système cyber-physiques, ordonnancement multiprocesseur, ordonnancement cyclique, graphes de flots de données statiques, architectures hétérogènes, pipeline logiciel, débit maximal.

**Résumé:** Les systèmes cyber-physiques sont des systèmes distribués qui intègrent un large panel d'applications logicielles et de ressources de calcul hétérogènes connectées par divers moyens de communication (filaire ou non-filaire). Ces systèmes ont pour caractéristique de traiter en temps-réel, un volume important de données provenant de processus physiques, chimiques ou biologiques. Une des problématiques rencontrée dans la phase de conception des systèmes cyber-physiques est de prédire le comportement temporel des applications logicielles. Afin de répondre à cette problématique, des stratégies d'ordonnancement statique sont nécessaires. Ces stratégies doivent tenir compte de plusieurs contraintes, notamment les contraintes de dépendances cycliques induites par l'exécution des boucles de calculs spécifiées dans les programmes logiciels ainsi que les contraintes de ressource et de communication inhérentes aux architectures matérielles de calcul. En effet, les boucles étant l'une des parties les plus critiques en temps d'exécution pour plusieurs applications de calcul intensif, le comportement temporel et les performances optimales des applications logicielles dépendent de l'ordonnancement optimal des structures de boucles spécifiées dans les programmes de calcul. Pour prédire le comportement temporel des applications logicielles et fournir des garanties de performances dans la phase de conception au plus tôt, les stratégies d'ordonnancement statiques doivent explorer et exploiter efficacement le parallélisme embarqué dans les patterns d'exécution des programmes à boucles intensives tout en garantissant le respect des contraintes de ressources et de communication des architectures de calcul. L'ordonnancement d'un programme à boucles intensives sous contraintes ressources et communication est un problème complexe et difficile. Afin de résoudre efficacement ce problème, il est indispensable de concevoir des heuristiques. Cependant, pour concevoir des heuristiques efficaces, il est important de caractériser l'ensemble des solutions optimales pour le problème d'ordonnancement. Une solution optimale pour un problème d'ordonnancement est un ordonnancement qui réalise un objectif optimal de performance. Dans cette thèse, nous nous intéressons au problème d'ordonnancement des programmes à boucles intensives sur des architectures de calcul multiprocesseurs hétérogènes sous des contraintes de ressource et de communication, avec l'objectif d'optimiser le débit de fonctionnement des applications logicielles. Pour ce faire, nous utilisons les modèles de flots de données statiques pour décrire les structures de boucles spécifiées dans les programmes de calcul et nous concevons des stratégies d'ordonnancement périodiques sur la base des propriétés structurelles et mathématiques de ces modèles afin de générer des solutions optimales et approximatives d'ordonnancement.

Université Paris-Saclay
Espace Technologique / Immeuble Discovery
Route de l'Orme aux Merisiers RD 128 / 91190 Saint-Aubin, France