Design and Analysis for Multi-Clock and Data-Intensive Applications on Multiprocessor Systems-on-Chip

Abdoulaye Gamatié

Numéro d'ordre : 40917

Université Lille 1 Sciences et Technologies

HABILITATION À DIRIGER DES RECHERCHES (HDR)

Discipline : Informatique

par

ABDOULAYE GAMATIÉ

DESIGN AND ANALYSIS FOR MULTI-CLOCK AND DATA-INTENSIVE APPLICATIONS ON MULTIPROCESSOR SYSTEMS-ON-CHIP

HDR soutenue le 15 novembre 2012, devant le jury suivant :

Jean-Luc Dekeyser	Professeur, Université de Lille 1	Examinateur
Rajesh K. Gupta	Professeur, Californie – San Diego, USA	Rapporteur
Nicolas Halbwachs	Directeur de Recherche CNRS, Verimag/Grenoble	Rapporteur
Axel Jantsch	Professeur, Royal Institute of Tech. – Stockholm, Suède	Examinateur
Paul Le Guernic	Directeur de Recherche INRIA, Rennes	Examinateur
Patrice Quinton	Professeur, ENS Cachan	Rapporteur

UNIVERSITÉ DE LILLE 1 – SCIENCES ET TECHNOLOGIES
LIFL - UMR 8022 - Cité Scientifique, Bât. M3 - 59655 Villeneuve d'Ascq Cedex

DESIGN AND ANALYSIS FOR MULTI-CLOCK AND DATA-INTENSIVE APPLICATIONS ON MULTIPROCESSOR SYSTEMS-ON-CHIP

ABDOULAYE GAMATIÉ

Habilitation Thesis
November 2012

In Memoriam of Boubakar.

To Leila and Sarah.

ABSTRACT

With the increase in the integration of functions, modern embedded systems have become very smart and sophisticated. The typical examples of this tendency are last generation mobile phones, which provide users with a large panel of facilities for communication, music, video display, built-in camera, Internet access, etc. These facilities are achieved by applications processing huge amounts of information, referred to as *data-intensive applications*. Such applications are also characterized by *multi-clock* behaviors since they often include components operating at different activation rates during execution.

Embedded systems often have real-time constraints. For instance, in a video processing application, there are usually rate or deadline constraints imposed to image display. For this purpose, execution platforms must provide the required computational power. Parallelism plays a central role in the answer to this expectation. The integration of multiple cores or processors on a single chip, referred to as *multiprocessor systems-on-chip* (MPSoCs) is a key solution to provide applications with sufficient *performance* at lower energy costs for execution. In order to find a good compromise between performance and energy consumption, resource *heterogeneity* is exploited in MPSoCs by including processing elements with different characteristics. For example, core processing units are combined with accelerators such as graphic processing units or field-programmable gate arrays. Besides heterogeneity, *adaptivity* is another important feature of modern embedded systems. It enables to manage in a flexible manner the performance parameters w.r.t. variations of system environment and execution platform.

In such a context, the development of modern embedded systems has become very complex. It raises a number of challenges considered in the contributions of this document, as follows:

- First, since MPSoCs are distributed systems, how can one successfully address their design correctness, such that the functional properties of deployed multi-clock applications can be guaranteed? This is studied by considering a correct-by-construction design methodology for these applications on multiprocessor platforms.
- Second, for data-intensive applications to be executed on such platforms, the next question is how can one adequately deal with their design and analysis, while fully taking into account their reactive nature and their potential parallelism?
- Third, when considering the execution of these applications on MPSoCs, the final question is how can one analyze their non functional properties (for instance, execution time or energy) in order to predict their execution performances? The answer to this question is expected to serve for the exploration of complex design spaces.

This document aims to answer the above three challenges in a pragmatic way, by adopting a model-based vision. For this purpose, it considers two complementary dataflow modeling paradigms: the *polychronous modeling* related to the synchronous reactive approach, and the *repetitive structure modeling* related to array-oriented data parallel programming. The former paradigm enables to reason on multi-clock systems in which components

interact without assuming the existence of a global or reference clock. The latter paradigm offers a powerful specification of the massive parallelism in a system as factorized repetitive dependency relations over multidimensional structures.

RÉSUMÉ

Avec l'intégration croissante de fonctions dans les systèmes embarqués modernes, ces derniers deviennent très intelligents et sophistiqués. Les exemples les plus emblématiques de cette tendance sont les téléphones portables de dernière génération, qui offrent à leurs utilisateurs un large panel de services pour la communication, la musique, la vidéo, la photographie, l'accès à Internet, etc. Ces services sont réalisés au travers d'un certain nombre d'applications traitant d'énormes quantités d'informations, qualifiées d'*applications de traitements intensifs de données*. Ces applications sont également caractérisées par des comportements multi-horloges car elles comportent souvent des composants fonctionnant à des rythmes différents d'activations lors de l'exécution.

Les systèmes embarqués ont souvent des contraintes temps réel. Par exemple, une application de traitement vidéo se voit généralement imposer des contraintes de taux ou de délai d'affichage d'images. Pour cette raison, les plates-formes d'exécution doivent souvent fournir la puissance de calcul requise. Le parallélisme joue un rôle central dans la réponse à cette attente. L'intégration de plusieurs cœurs ou processeurs sur une seule puce, menant aux systèmes multiprocesseurs sur puce (en anglais, *multiprocessor systems-on-chip – MPSoCs*) est une solution clé pour fournir aux applications des performances suffisantes, à un coût réduit en termes d'énergie pour l'exécution. Afin de trouver un bon compromis entre performance et consommation d'énergie, l'hétérogénéité des ressources est exploitée dans les MPSoC en incluant des unités de traitements aux caractéristiques variées. Typiquement, des processeurs classiques sont combinés avec des accélérateurs (unités de traitements graphiques ou accélérateurs matériels). Outre l'hétérogénéité, l'adaptativité est une autre caractéristique importante des systèmes embarqués modernes. Elle permet de gérer de manière souple les paramètres de performances en fonction des variations de l'environnement et d'une plate-forme d'exécution d'un système.

Dans un tel contexte, la complexité du développement des systèmes embarqués modernes paraît évidente. Elle soulève un certain nombre de défis traités dans les contributions de ce document, comme suit :

- tout d'abord, puisque les MPSoC sont des systèmes distribués, comment peut-on aborder avec succès la correction de leur conception, de telle sorte que les propriétés fonctionnelles des applications multi-horloges déployées puissent être garanties ? Cela est étudié en considérant une méthodologie de distribution "correcte-par-construction" pour ces applications sur plates-formes multiprocesseurs.

- Ensuite, pour les applications de traitement intensif de données à exécuter sur de telles plates-formes, comment peut-on aborder leur conception et leur analyse de manière adéquate, tout en tenant pleinement compte de leur caractère réactif et de leur parallélisme potentiel ?

- Enfin, en considérant l'exécution de ces applications sur des MPSoC, comment peut-on analyser leurs propriétés non fonctionnelles (par exemple, temps d'exécution ou énergie), afin de pouvoir prédire leurs performances ? La réponse à cette question devrait alors servir à l'exploration d'espaces complexes de conception.

Ce document vise à répondre aux trois défis ci-dessus de manière pragmatique, en adoptant une vision basée sur des modèles. Pour cela, il considère deux paradigmes complémentaires de modélisation flot de données : la *modélisation polychrone* liée à l'approche synchrone réactive, et la *modélisation de structures répétitives* liée à la programmation orientée tableaux pour le parallélisme de données. Le premier paradigme permet de raisonner sur des systèmes multi-horloges dans lesquels les composants interagissent, sans supposer l'existence d'une horloge de référence. Le second paradigme est quant à lui suffisamment expressif pour permettre la spécification du parallélisme massif d'un système.

FOREWORD

First steps in research. My very first experience in the exciting world of academic research dates back to my Master internship, started by the end of 1999, in the Ep-Atr group (leader: Paul Le Guernic), which later became Espresso group (leader: Jean-Pierre Talpin) at the IRISA lab. in Rennes. Both groups investigate the correct design of safety-critical applications by considering the synchronous reactive approach. I started my Ph.D. thesis in the Espresso group from October 2000. The defense was in May 2004. After that, I stayed in this group during one year and half as a research associate and assistant professor affiliated to Université de Rennes I. Within all this period, I addressed the design of avionic real-time applications with the synchronous programming language Signal. This coincided with several studies in Espresso, on the *polychronous semantic model of Signal* [163], considered as the reference model for this language nowadays. Note that the first studies about the Signal language have been initiated by Paul Le Guernic [159] in strong collaboration with Albert Benveniste. Another major contributor is Thierry Gauthier, who developed the first compiler of this language. For more details on the main contributors to Signal, the reader can refer to the historical notes available in `http://www-users.cs.york.ac.uk/~burns/papers/signal.pdf`.

My work regarding Signal-based design and analysis has been supervised by Paul Le Guernic and Thierry Gautier, who have been a constant source of inspiration in my research over the past twelve years. My results have contributed to an existing rich literature on how asynchronous mechanisms, e.g., regarding communication, can be described and analyzed using the synchronous reactive approach.

Maturity era. In September 2005 when I joined the DaRT team-project led by Jean-Luc Dekeyser in Lille, this team was only two years old. Its main research topic was about the codesign of data-intensive systems-on-chip (SoCs) with a focus on the following issues: *i)* definition of a UML profile[1] for SoC comodeling, *ii)* compilation devoted to data-parallel constructs for an efficient mapping on multiprocessor platforms, and *iii)* simulation techniques mainly in SystemC. On my arrival in the team, I first occupied a Post-doc position for one year. Then, I obtained a full-time position as a CNRS Research Scientist. In the same period, together with Éric Rutten[2], we initiated a connection between the research topic of DaRT and the synchronous reactive approach. Our motivation was to exploit the complementarity between the techniques inherent to both approaches. The early work of Smarandache *et al.* [219] on a connection of the Alpha and Signal languages was one important inspiration to us.

For seven years, I have been investigating, with colleagues, this complementarity in order to provide a satisfactory answer to the crucial design issues of modern embedded systems. The obtained results are currently relevant enough to contribute to the maturity of my research activity. I am particularly grateful to Jean-Luc Dekeyser, Éric Rutten and Pierre Boulet for

1 Today integrated in the Marte standard profile [194] of the Object Management Group.
2 Éric Rutten joined the DaRT group in 2004 and moved to INRIA Grenoble in 2006. We both belong to the synchronous approach community.

our fruitful collaboration on this topic. Last but not least, I am deeply indebted to all my Ph.D. and Master students, and Post-doc colleagues, who actively contributed to the different results summarized in this document.

On the other hand, my decision to defend this Habilitation thesis coincides with an important re-organization[3] of the DaRT group, around new research topics currently under discussion. I such a context, I am planning to join a new research group in another laboratory, in accordance with my research perspectives highlighted at the end of this document.

Rationale of this synthesis document. This document is a summary of my scientific contributions from my Post-doc to nowadays. It aims to show in a coherent way how my research activity on the high-level design and analysis of embedded systems has evolved from multi-clock distributed applications to performance-demanding applications, such as data-intensive applications.

To address the former applications, I considered the *polychronous modeling* paradigm, related to the synchronous reactive approach, to show how they can be safely designed on globally asynchronous locally synchronous (GALS) platforms, so that the correctness of their functional behaviors is ensured.

To deal with the latter applications, I combined the polychronous model with the *repetitive structure modeling* paradigm, which is suitable for describing massively parallel computations and platforms. Beyond the correctness of the functional behaviors, I have been also exploring adequate ways to find the best performance and energy/power trade-offs for these performance-demanding applications. Here, the target execution platforms are multiprocessor systems-on-chip (MPSoCs)

How to consider the contents of this document. All results reported in the contribution chapters, i.e., Chapters 2, 3 and 4, have been already published in peer-reviewed journals and conferences. The presented material therefore comes in major part from the related publications, which are explicitly mentioned as *margin notes* throughout the text. In that way, the reader can easily find more details about each summarized contributions. Most of my papers that occur as references in this document are available online via the websites of their editors. Finally, in order to easily assess my achievements, an *executive summary* is provided at the end of each contribution chapter.

3 The LIFL lab. (Computer Science Laboratory of Lille, `http://www.lifl.fr/`) to which DaRT belongs and the LAGIS lab. (focusing among others on continuous and discrete event dynamical systems, system safety engineering and vision and image processing in Lille, `http://lagis.ec-lille.fr/`) are going to be merged in order to create a unique laboratory in the North of France, where the complementarity of their covered research areas will be better promoted.

ACKNOWLEDGMENTS

First of all, I am very grateful to Prof. Rajesh K. Gupta from UC San Diego (USA), Dr. Nicolas Halbwachs from Verimag/CNRS (France) and Prof. Patrice Quinton from ENS Cachan (France) for having kindly accepted to serve as reviewers on my Habilitation defense committee, and for the reports they wrote on my work.

I would like also to thank warmly Prof. Jean-Luc Dekeyser from University of Lille 1 (France), Prof. Axel Jantsch from Royal Institute of Technology (Sweden) and Dr. Paul Le Guernic from Inria (France), for their kind participation to my Habilitation defense committee as examiners, and for the enlightened discussions we had during the defense.

A major part of the contributions presented in this document have been obtained in collaboration with colleagues from the following research groups: Aoste group in Sophia-Antipolis (France), DaRT group in Lille (France), Espresso group in Rennes (France), Electronic Systems group in TU/Eindhoven (The Netherlands), Fermat group at Virginia Tech (USA) and Sardes group in Grenoble (France). These colleagues are too many to name all. Many thanks to all of them!

I want to express especially my gratitude to my colleagues from the Espresso and DaRT research groups for their insightful comments on the draft version of this document and for their very useful support in the organization of my Habilitation defense.

Finally, many thanks to my family for its constant and precious support.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

In a keynote at the Euromicro conference on Digital System Design (DSD) in 2006, Duranton [75] raised the following major challenges regarding high performance embedded systems: the *management of parallelism and time requirements, the co-modeling for application mapping on execution platforms, and correctness and efficiency of designs*. These challenges must be addressed within stringent time-to-market and low cost constraints. This document brings some answer elements to address such challenges. It presents a summary of my contributions since September 2004 on the design and analysis for multi-clock and data-parallel applications on multi-processor embedded systems. These works have been achieved successively at the IRISA (in Rennes) and LIFL (in Lille) labs, associated with Inria. Most of the results have been obtained in collaborations with colleagues from different labs and countries. The presented material partly comes from our common publications. I will use "I" or "We" nominative cases when appropriate to present the results.

IRISA stands for Institut de Recherche en Informatique et Systèmes Aléatoires (`http://www.irisa.fr/english/home.html`)

LIFL stands for Laboratoire d'Informatique Fondamentale de Lille (`http://www.lifl.fr/?languageId=1`)

The remainder of this introductory chapter is organized as follows: Section 1.1 discusses some major trends in embedded system design according to which my research is driven; Section 1.2 summarizes my scientific contributions; and Section 1.3 describes the outline of the document.

1.1 TRENDS IN EMBEDDED SYSTEM DESIGN

Embedded systems are special-purpose computer systems combining software and hardware components that are subject to external constraints coming from environment and execution platforms. Their implementation on chips, referred to as systems-on-chip (SoCs) makes them pervasive, ubiquitous and suitable in many modern applications. Examples are consumer electronics that currently propose very tempting electronic gadgets: digital cameras, GPS receivers and video player. Among these, the most emblematic products are probably mobile smart-phones providing an impressive access to communication and entertainment services and social networking. According to Parks Associates[1], the number of mobile phone users worldwide will reach 4.5 billion by 2013. At the same time, the percentage of the marketing budget will increase to 150% according to Telefonica O2[2]. Even emerging and developing economies will significantly contribute to this expected growth. Beyond their wide adoption in consumer electronics, embedded systems are also present in the following application fields [76]: automotive electronics such as in-vehicle entertainment systems; civil and military defense such as radar, sonar, satellite systems and weather systems; medical electronics such as surgical systems, imaging and diagnosis equipments; computational science for simulating complex physical phenomena such as climate modeling, seismic waves, behaviors of biological systems; and business information processing from databases.

1 `http://www.itfacts.biz/45-bln-mobile-users-by-2013`
2 `http://www.cellular-news.com/story/34048.php`

1.1.1 *High function integration and data volumes*

With the high *integration* of functions, embedded systems have become very smart and sophisticated. Last generation mobile phones provide users with a large panel of facilities for communication, music and video players, built-in camera, Internet access, etc. Another example is the Sony PlayStation, which integrates many similar functions. Future embedded multimedia systems are expected to keep on integrating more functions.

All these facilities within a single system lead to a processing of huge amounts of information by the system. For instance, a mobile phone can contain gigabytes of video, photo and music data files to process. Many of the application domains mentioned previously include *data-intensive processing*: applications operate on large data sets [213] or streams of data where the processing mostly consists of data read/write and data manipulation. The amount of manipulated data is expected to double every two years in the future in these domains [47]. A number of common characteristics can be observed in data-intensive applications:

- the sets of data are represented by multidimensional data structures, e.g., multidimensional arrays, where dimensions express metrics such as time, space, frequency, temperature, magnetic field, etc. The information stored in these data structures are accessed either point-wise via array indexes, or block-wise via monolithic data subsets.
- the computation of output data is achieved by applying operations such as filters in multimedia signal processing to input data, independently from the order in which these data are treated by operations. The set of output data is often smaller than the set of input data.

When applications get access to data in the associated data-structures in a regular and predictable way, they are often referred to as *regular data-intensive applications*. Those characterized by unpredictable memory access patterns, control structures and network transfers are referred to as *irregular data-intensive applications*. They typically manipulate pointer-based data structures, e.g., linked lists, graphs and trees.

The above trends to increase the integration of functions and the amount of data to be processed by applications inevitably amplifies the complexity of the development for modern embedded systems. In addition, it strongly calls for scalable design solutions.

1.1.2 *Parallelism for high-performance and power-efficiency*

Embedded systems often have real-time constraints, e.g., in a video processing application, there are usually rate and deadline constraints imposed to image display. Execution platforms must provide the required computational power. Parallelism plays a key role in the answer to this expectation [87]. Instead of accelerating the clock frequency of each new processor generation as in a near past, technology providers such as Intel [39] now opt for integrating multiple cores or processors on a single chip (by densifying the number of transistors), referred to as *multiprocessor systems-on-chip* (MPSoCs) [241] in order to obtain better execution performances. As a result, the traditionally adopted von Neumann model of computer architecture and sequential programming models should be revisited, or even discarded.

In addition to processors or cores, MPSoCs include components such as accelerators for video and audio signal processing, communication means, e.g., memories and their interconnects, and peripherals (see Figure 1). They are adopted in the PlayStation 3, which uses a Cell Broadband Engine (CBE) [148] composed of one PowerPC core and eight synergistic cores, i. e., single instruction multiple data processing units, dedicated to data-intensive processing. According to predictions of the International Technology Roadmap for Semiconductors (ITRS) [139], the number of processing cores and memory size in portable equipments will increase by a factor of 6 in the next eight years to reach around eight hundred processors.

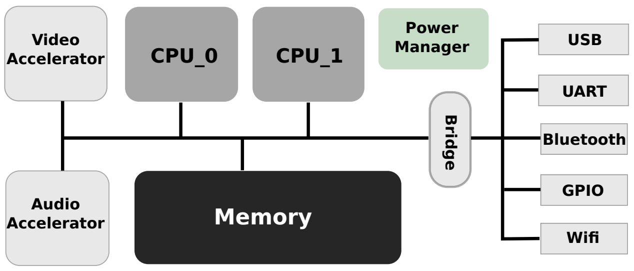


Figure 1: A simplified model of MPSoC.

Notice that the aforementioned performance enhancement of computer architectures mostly concerns computing resources. While the latency of computations achieved by multiple CPUs is reduced, it is not necessarily the case for the overall temporal performance of systems due to inadequate memory access time and data bandwidth in communications, via a shared memory. The difference between CPUs and memory cycle times is currently about a factor of 1000, referred to as *memory wall* [152].

The high-performance requirements of embedded systems go together with the need to keep their power/energy consumption at a minimum level (AA battery lifetime is *de facto* a metric in portable embedded systems today). Especially when the systems are embedded in portable devices whose autonomy strongly depends on batteries. In order to find a good compromise between performance and energy consumption, exploiting *heterogeneity* in multiprocessor systems seems to be promising. Such systems include processing elements with various characteristics, e.g., architectures with processors running at different frequencies, or architectures combining core processing units (CPUs) and accelerators such as graphic processing units (GPUs) or field-programmable gate arrays (FPGAs). Accelerators allow to save energy when executing performance-demanding pieces of code in data-intensive applications. For instance, the Xbox 360 (Slim) console[3] of Microsoft, which integrates CPU and GPU components on a chip, reduces the power consumption by more than 20%. We refer to systems with the same type of processing elements as *homogeneous* systems.

As an overall picture, Table 1 indicates next generation embedded applications with typical performance and power requirements obtained from [78]. A large part of these applications concerns data-intensive computation.

1.1.3 *Adaptivity in embedded systems*

Adaptivity is more and more desired in embedded systems for several reasons. First, the ability to adapt to environment variations becomes very important. For instance, a video-surveillance embedded system for street observation adapts its image analysis algorithms according to factors like the

3 http://www.xbox.com

Table 1: Performance/power needs of next generation applications (Gops, mW and kW refer to *Giga operations per second*, *milliwatt* and *kilowatt* resp.) [78].

Field	**Application**	**Performance**	**Power**
Mobile and Wireless Computing	Speech recognition, video compression, network coding & encryption	10–40 Gops	100 mW
High-Performance Computing	Computational fluid dynamics, molecular dynamics, life sciences, oil and gas, climate modeling	100–10000 Gops	100–1000 kW
Medical Imaging and Equipments	3D reconstruction, image registration and segmentation, battery-driven health monitoring	1–1000 Gops	100 mW–100 W
Automotive	Lane, collision and pedestrian detection, driving assistance systems	1–100 Gops	20–500 W
Home and Desktop Applications	Gaming physics, ray tracing, CAD tools, EDA tools, web mining	10–1000 Gops	20–500 W
Business	Portfolio selection, smart cameras, asset liability management	1–1000 Gops	1–100 W

human activity (crowded place or not), luminosity (day or night) or the weather. Some video-processing systems need to adapt their data processing according to the consumption and production rates of input and output information.

On the other hand, we can also observe the increasing *variability of hardware* architectures in embedded systems [198]. Indeed, with the need to provide high-performance via parallelism, the high integration of transistors on a chip (e.g., more than two billions in the Intel 4-cores Tukwila) imposes extreme chip manufacturing technologies such as the Intel 22 nm Tri-Gate announced in 2011. Unfortunately, at this evolution, the percentage of defects in chip manufacturing grows. As a result, a chip may not always provide the full performance guarantees as expected, i.e. variability effects. Embedded systems must adapt to this fact. Another limitation of current chips comes from the enabled thermal dissipation of electronic components, which imposes only a partial use of execution capacities. Depending on required computation power, an embedded system must adapt so as to exploit the strict necessary chip resources. Such an idea is advocated by the dark silicon chips [81] in which the number of transistors is greater than the one they can supply in power.

Finally, we observe that *reconfigurable computing*, which has been studied for several years [232], is getting mature nowadays. With the increasing popularity of FPGA-based design, it is a potential solution to the implementation of adaptive embedded systems.

1.2 CONTRIBUTIONS: CORRECT AND EFFICIENT MPSOC DESIGN

Position statement.

There is an enthusiastic debate on the nature of future embedded systems, in which massive parallelism will be a key feature. This evolution suggests a re-visitation of current design practice for an adequate outcome. Many prominent prospective reports [76] [12] [152] call for new programming models that fit well the design challenges of future embedded systems. To fill this demand, I strongly believe there is real opportunity to look back on existing models and build on top of them the expected golden programming models. I advocate the use of the well-established dataflow computing model [149] [73] [238], via the combination of *i)* the multi-clock synchronous dataflow model, supported by the synchronous programming languages [29], and *ii)* the affine multidimensional dataflow model, supported by the so-called repetitive structure modeling (RSM) based on the Array-oriented language Array-OL [72]. This combination favors a very interesting high-level design of embedded systems, by providing:

- a rich expressivity for modular description of concurrency and massive parallelism in systems, as well as behavioral constraints imposed by execution platforms and environment,
- a connection to a set of verification/analysis techniques and tools for design assessment
- an access to efficient compilation techniques for loop and control optimized code generation towards distributed implementation platforms, e.g. MPSoCs.

Starting from studying the applicability of the so-called *polychronous* model [163] (i.e., a multi-clock dataflow model that does not assume *a priori* the existence of a global clock in a system) to the safe design of distributed embedded systems, I have been investigating the integration of this model together with RSM, in a data-intensive MPSoC codesign framework [106]. The obtained results show that such an approach can provide a rapid, flexible and relevant way to assess multiple design alternatives, w.r.t. correctness, real-time constraints, performances and energy. This is crucial for a successful and cost-effective design of future embedded systems.
This promising outcome calls for continuing the development of our high-level approach by taking into account other mainstream features of future embedded systems, in particular their adaptivity or reconfiguration needs. My on-going research already explores this direction, which I plan to study more in my future research, with a shift to lower abstraction levels in order to finely address platform details.

From the trends presented in Section 1.1, it appears that with the technological advances, including the increasing integration of transistors in chips and reconfigurable computing, embedded systems become extremely sophisticated integrated systems. This obviously makes their design very complex. Unfortunately, this evolution happens without taking care of the limitations of existing design and verification tools. Within such a context, my contributions aim to the design of MPSoCs where correctness and performance can be addressed at a high abstraction level to provide flexibility and cost-effectiveness. The advocated solution targets adaptive embedded systems achieving highly parallel regular applications with temporal constraints. The considered execution hardware platforms can be either homogeneous or heterogeneous.

Figure 2 summarizes my research activities since 1999, according to three system design layers: *software application*, *software/hardware interface* and *hardware platform*. My contributions only concern the first two layers. In the software application layer, design and analysis issues are addressed by mainly focusing on application functionality. In the software/hardware interface layer, the interaction between software applications and hardware execution platforms is dealt with by explicitly taking into account features of both parts.

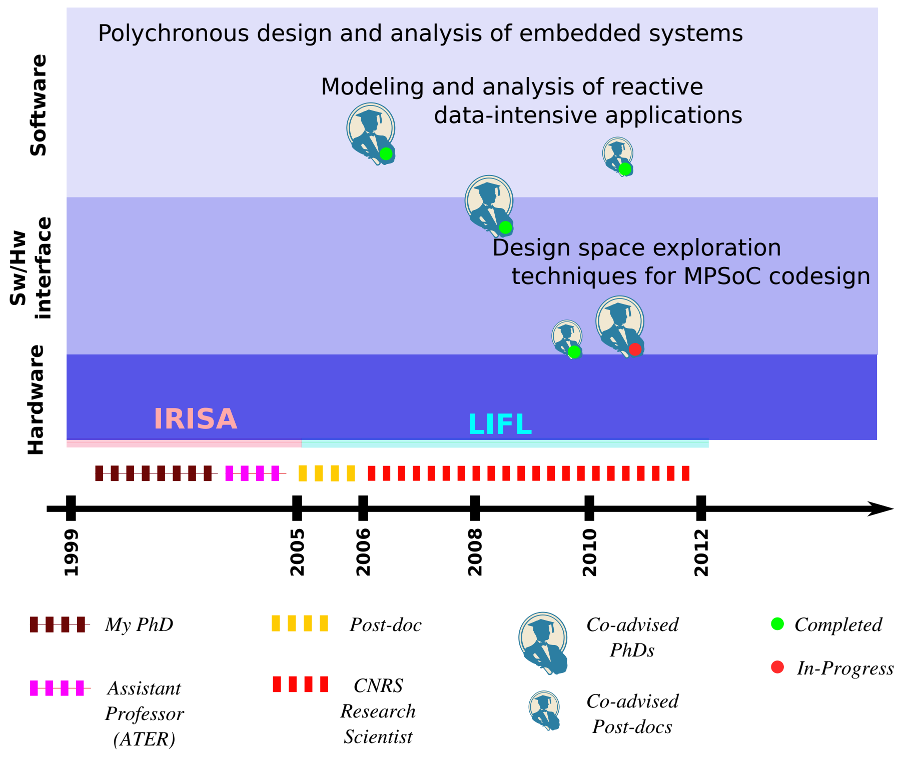

Software
Sw/Hw interface
Hardware
Polychronous design and analysis of embedded systems
Modeling and analysis of reactive data-intensive applications
Design space exploration techniques for MPSoC codesign
IRISA
LIFL
1999
2005
2006
2008
2010
2012
My PhD
Post-doc
Co-advised PhDs
Completed
In-Progress
Assistant Professor (ATER)
CNRS Research Scientist
Co-advised Post-docs

Figure 2: A visual summary of my contributions, according to system design layers over the time represented by the horizontal line at the bottom (the French acronym ATER denotes "*Attaché Temporaire d'Enseignement et de Recherche*" and is equivalent to Assistant Professor position).

The first part of my contributions, devoted to *polychronous design* in Signal, was initiated in 2000 during my PhD at IRISA lab and has been continued until now. It concerns the safe design of distributed embedded systems and an improved static analysis of multi-clocked application specifications for optimized automatic code generation. The second part of the contributions has started in 2005 after joining the LIFL lab (first as Post-doc, then as CNRS Research Scientist). It deals with the co-modeling of data-intensive applications on MPSoCs platforms. A modeling paradigm mixing data-parallel computations, adaptive behaviors and temporal constraints has been studied. It mainly relies on the polychronous modeling and the *repetitive structure modeling*. The last part of the contributions is the most recent, i. e., from

CNRS stands for Centre National de la Recherche Scientifique

2008. Here, I have been building design space exploration techniques on top of previous parts in order to tackle the correctness and performance issues in MPSoC design.

During my research activities, I have participated to the advisory of three PhD. students: two already defended and one ongoing (Figure 2 only shows their starting period according to the horizontal temporal line). I also mentored two Post-doc fellows. In addition, notice that I have supervised several Master students, which are not reported in Figure 2.

The next sections summarize my main contributions presented in this document.

1.2.1 *Polychronous design and analysis of systems*

Embedded systems are usually composed of elements operating at various rhythms, e. g., different sensor and actuator activation rates, communication bus and processors frequencies. From the overall point of view, this naturally leads to *multi-clock embedded systems* in which multiple activation clocks are related by constraints, capturing the interaction (synchronization and communication) expected between system elements. MPSoCs are one example of multi-clock and distributed embedded systems. The *polychronous modeling* paradigm [163], adopted by the Signal language [26] [92], enables to describe such systems without necessarily assuming a reference clock. The interaction of different parts of a system is captured via precedence and coincidence relations between event occurrences.

The next paragraphs summarize my main contributions on the polychronous design of embedded systems.

DESIGN OF MULTI-CLOCK DISTRIBUTED EMBEDDED SYSTEMS. Since my PhD thesis defended in May 2004 in the Espresso group of IRISA (in Rennes), I have continued to investigate the design of multi-clock embedded applications on architectures such as globally asynchronous locally synchronous (GALS) ones [53] [187]. I consider the polychronous design model. These studies are carried out in collaboration with colleagues from Espresso. With Thierry Gautier and Paul le Guernic, I addressed methodological aspects for system design by providing a set of design rules based on correct Signal program distribution method and a library of architecture components. Such components include various asynchronous communication and synchronization mechanisms that I modeled in Signal during my PhD thesis. They have been integrated since then in Polychrony [4], the development environment of Signal. With Thierry Gautier, Jean-Pierre Talpin and Christian Brunette, we also proposed a few pragmatic extensions for Signal in order to facilitate control-oriented design as in Mode Automata [179]. This extension has been implemented in the Signal-Meta environment (SME), which is a front-end of Polychrony in the Eclipse environment, and based on Model-Driven Engineering (MDE) technologies.

Espresso stands for Environnement de spécification de programmes réactifs synchrones (`http://www.inria.fr/en/teams/espresso`)

STATIC ANALYSIS OF POLYCHRONOUS PROGRAMS. Beyond design aspects, I have been involved in a few studies about the static analysis of polychronous specifications, which is crucial for correctness and optimization of automatically generated code. In particular, I focused on the verification of numerical properties, which are not fully addressed with the widely adopted Boolean abstractions in compilers of synchronous languages. This

4 `http://www.irisa.fr/espresso/Polychrony`

research topic originated from early discussions with Paul Le Guernic during my Master internship in 1999 within the Ep-Atr group of IRISA. These studies were conducted in collaboration with Thierry Gautier and Loïc Besnard of Espresso group, by considering an approach based on interval decision diagrams (IDDs) [223]. Then, more recently with Laure Gonnord from the DaRT group of LIFL, and Sandeep Shukla and Bijoy Jose from the Fermat lab at Virginia Tech (VA, USA), we adopted an alternative solution based on satisfiability modulo theory (SMT) [69].

Ep-Atr stands for Environnement de programmation d'applications temps réel (`http://www.inria.fr/en/teams/epatr`)

DaRT stands for Apports du parallélisme données au temps réel (`http://www.inria.fr/en/teams/dart`)

The above investigations about the polychronous design of embedded systems have been carried out mainly in the Polychrony environment. They strongly favored a personal experience on Signal programming, which I restituted in a book [92], edited by Springer in 2010.

1.2.2 *Design of reactive data-intensive applications*

As the parallelism level in both applications and execution platforms is growing significantly, massively parallel embedded systems will be very present in several application domains. This calls for adequate design concepts offering an efficient and expressive description of the massive parallelism inherent to data-intensive applications implemented on multiprocessor platforms. The *repetitive structure modeling* (RSM) paradigm [115] based on the Array-OL formalism [72], expresses the massive parallelism as factorized repetitive dependency relations over multidimensional structures. Thanks to these features, it has been adopted in the UML/Marte standard profile [194], for modeling and analysis of real-time and embedded systems.

An overview of my main contributions on the modeling and analysis of reactive data-intensive applications is given below.

MULTI-PARADIGM MODELING AND ANALYSIS APPROACH. From my Post-doc started in September 2005 for one year, I have joined the DaRT group of LIFL and Inria (Lille), which investigates the design of embedded data-intensive applications on massively parallel architectures. My motivation was to explore the applicability of the polychronous model in this specific application field. I closely collaborated with Éric Rutten, Jean-Luc Dekeyser and Pierre Boulet to understand the link between the polychronous and RSM modeling paradigms. In order to benefit from the respective capabilities of both paradigms, we studied a translation of RSM specifications into synchronous dataflow programs [102]. This has been achieved in the context of the PhD thesis of Huafeng Yu [244] (co-advised with Éric Rutten and Jean-Luc Dekeyser). The defined translation offers a bridge according to which a designer can specify the potential parallelism of a given application and produce a corresponding synchronous model analyzable with the synchronous technology. Among the properties of interest in RSM models, are the absence of causal cycles, absence of multiple assignments to array variables, etc. They are safely addressed with compilers of synchronous languages. However, the main limitation of our translation is its scalability. As a solution, with Pierre Boulet, we proposed a component-based abstraction by using loop transformations to mitigate the massive parallelism expressed in RSM. This promotes a modular design from which data dependencies can be better managed. Parts of these results are covered by the five-months Post-doc of Mohamed Fellahi, co-advised with Pierre Boulet.

FROM STATIC TO DYNAMIC MODEL FOR CODESIGN. The data dependencies expressed within RSM only define a static partial ordering in application behaviors. This is clearly not expressive enough to describe the dynamics related to control flows or temporal aspects. During the PhD thesis of Huafeng Yu, we studied the refinement of a preliminary extension [156] of RSM with modes and finite state machines, again inspired by Mode Automata. We proposed a generic extended model usable in SoC co-design levels [70]: software application, hardware architecture, association of both. It is one of the few propositions in literature mixing multidimensional dataflow with control. It has been experimented in discrete control synthesis for multimedia applications and hardware accelerators generation for reconfiguration in FPGAs. On the other hand, in order to describe temporal aspects in RSM, we proposed to refine data dependency specifications with abstract clock constraints, expressed with the Marte Time concepts and the *clock constraint specification language* (CCSL) [16]. These constraints explicitly capture suitable information about environment and execution platform properties of systems. The result are clock traces reflecting simulation scenarios for a system, which serve for an easy and rapid design assessment. This work has been carried out in the context of the PhD thesis of Adolf Abdallah (co-advised with Jean-Luc Dekeyser) and a collaboration with the colleagues from the Aoste group of Inria and I3S (Sophia Antipolis).

Aoste stands for Modèles et méthodes pour l'analyse et l'optimisation des systèmes temps réel embarqués (http://www.inria.fr/en/teams/aoste)

I3S stands for Laboratoire d'Informatique, Signaux et Systèmes de Sophia Antipolis (http://www.i3s.unice.fr/I3S/presentation.en.html)

Most of the above works have been experimented in the Gaspard2[5] design environment, which uses Marte for the codesign of data-intensive SoCs.

1.2.3 *Design space exploration for MPSoC codesign*

Relevant design properties of embedded systems include functional correctness, temporal performance, memory size and energy consumption. Techniques such as physical prototyping and simulation are still widely used for design assessment. However, with the increasing complexity of embedded systems, they are insufficient to explore large design spaces. Leveraging high abstraction levels is probably the key ingredient for overcoming this limitation. Indeed, higher level approaches are fast, cost-effective and permit relevant analysis of complex design spaces. The desired design space exploration (DSE) solutions must identify adequate parallelism levels, configuration parameters and hardware/software mappings, w.r.t. behavioral correctness, best execution performance, memory and energy consumption. In order to define such solutions for MPSoCs, I adopt two complementary approaches, highlighted in the next.

DSE FOR EFFICIENT DATA TRANSFER AND STORAGE. From December 2009, I have been collaborating with Pierre Boulet and Rosilde Corvino, a former Post-doc in DaRT group (co-advised with Pierre Boulet) on the definition of DSE for data transfer and storage micro-architectures [63], which include communication structures and memory hierarchies. Rosilde is research scientist and project manager at TU/e in Eindhoven (The Netherlands) since December 2010. We keep on working on this topic together with her colleagues in Eindhoven.

Today, the optimization of communication structures, memory hierarchy and global synchronizations in embedded systems is a time consuming and

5 "Gaspard2" denotes the second version of an environment dedicated to *Graphical Array Specification for PARallel and Distributed computing* – http://www.gaspard2.org

error-prone process. As an answer, we proposed an electronic system level framework to explore the best communication and synchronization configurations of data-parallel applications. In Gaspard2, this enables to assess various mappings of RSM application models onto MPSoC architectures. Most of existing works improving hardware synthesis with loop transformations, optimize the loop iteration scheduling, reduce the redundant memory traffic and improve the synthesis of computing data path only for *single nested loops*. Our solution enables to explore different loop transformations for applications with *multiple communicating nested loops* [66]. From these transformations, it infers architecture template customizations. In order to make efficient the implementation of our analysis, we used the abstract clock notion of the synchronous reactive model to capture scheduling and mapping information of repetitive tasks, i. e., loops, when mapped onto customized architecture templates. The exploration process is performed through a genetic algorithm.

EXPLORING PARALLELISM LEVEL IN HARDWARE/SOFTWARE MAPPING. Following the work initiated during the PhD thesis of Adolf Abdallah [2], we are developing an abstract clock-based framework for analyzing adaptive MPSoC systems. This work is conducted in the PhD thesis of Xin An started in October 2010 (co-advised with Éric Rutten in the Sardes group of Inria Grenoble).

Sardes stands for Architecture de systèmes réflexifs pour les environnements distribués (http://www.inria.fr/en/teams/sardes)

We use a multi-clock modeling for combined software, hardware and environment specifications to overcome the design validation issues [93]. An application is represented according to the polychronous model by specifying event occurrences with their precedence relations. Then, we study how to check possible temporal constraints imposed by an environment on applications by exploiting affine clocks of Signal. We analyze different design scenarios of applications mapping and scheduling on MPSoC platforms via abstract clock traces representing system simulations. From these traces, behavioral correctness can be checked. Execution times can be also determined for performance evaluation. In addition, the ability to easily reproduce such kinds of traces for different processor frequency values favors a qualitative reasoning about energy consumption. All this is also made possible in presence of adaptive system behaviors, including frequency changes during execution and task migrations [15]. A major advantage of this approach is that it offers a simple and fast alternative to explore and reduce complex design spaces before applying physical prototyping and simulation techniques. It is an ideal complement to these techniques.

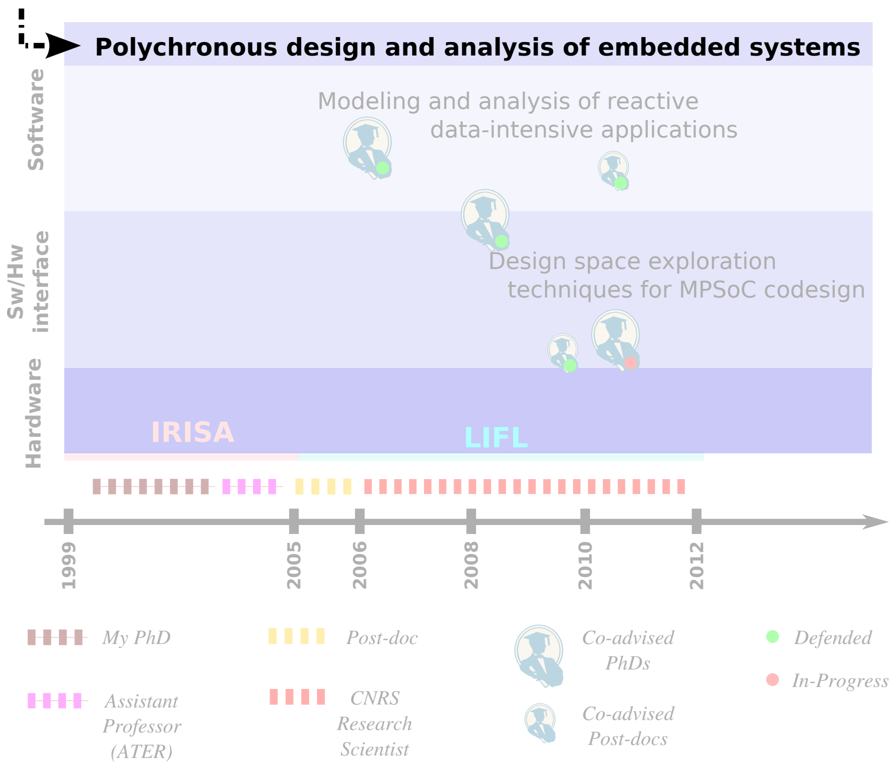
The above two works on design space exploration techniques for MPSoCs have been implemented in two prototype tools, available on demand.

1.3 OUTLINE OF THE DOCUMENT

The remaining of this report is organized as follows: Chapter 2 presents our works on the correct design of multi-clocked distributed embedded systems by using the polychronous design model associated with the Signal language; Chapter 3 summarizes the modeling and analysis of reactive data-intensive applications by combining the multidimensional repetitive structure modeling and the multi-clock synchronous modeling paradigms; Chapter 4 reports our recent works defining two approaches for the analysis and design space exploration of data-parallel applications on MPSoCs; finally, Chapter 5 gives the conclusions and draws future research directions.

2 POLYCHRONOUS DESIGN OF EMBEDDED SYSTEMS

The contributions presented in this chapter cover a range of works that I have been involved in since my PhD defense in May 2004, on the polychronous design of distributed embedded systems (see Figure 3). They have been obtained mostly in the context of a collaboration with colleagues from Espresso, my former research group at IRISA in Rennes. Another part of these contributions, mainly on static analysis of polychronous programs, comes from recent collaborations with Laure Gonnord from LIFL in Lille and Sandeep Shukla's group from Virginia Tech in USA. These contributions are part of a strong personal motivation to understand thoroughly the polychronous modeling, to construct typical representative examples illustrating its capabilities for the safe design of multi-clock distributed embedded systems, and to bring it to a wide audience, and most importantly for educational purpose.



Polychronous design and analysis of embedded systems
Software
Sw/Hw interface
Hardware
Modeling and analysis of reactive data-intensive applications
Design space exploration techniques for MPSoC codesign
IRISA
LIFL
1999
2005
2006
2008
2010
2012
My PhD
Post-doc
Co-advised PhDs
Defended
In-Progress
Assistant Professor (ATER)
CNRS Research Scientist
Co-advised Post-docs

Figure 3: Specific contributions presented in the current chapter (the other contributions not exposed here are intentionally blurred).

The chapter is organized as follows: in Section 2.1, I give some motivations for polychronous design and introduce the main challenges addressed in my different contributions; in Section 2.2, I present an overview of my works on the design of distributed embedded systems with the polychronous model; in Section 2.3, I summarize my proposition on the usage of satisfiability modulo theory for a better static analysis of polychronous specifications; in Section 2.4, the pedagogical implication of these works is discussed; finally, in Section 2.5, I discuss the strengths, limitations and future directions to presented works. An executive summary is given, regarding the key points in my contributions highlighted in this chapter.

2.1 OVERVIEW OF MAIN CHALLENGES

MPSoCs have been adopted in consumer electronics to achieve high quality of service (QoS). They support advanced techniques allowing to change dynamically the frequency of processing w.r.t. to the voltage, i. e., dynamic voltage and frequency scaling (DVFS) [177]. The multiple clock domains resulting from local decisions to increase or decrease a processor frequency offer a flexible way to address a global performance/energy trade-off in systems. A similar observation is made at system level when designing embedded applications as functional blocks or modules, running concurrently on different computation nodes, for instance multi-rate tasks.

Globally asynchronous locally synchronous (GALS) architectures [53] [187] used to be an interesting implementation of such multiple clock domains systems. Each computation node in GALS holds its own clock providing a local (synchronous) vision of time. The GALS model is attractive for the design of multi-clock distributed systems thanks to its composability.

In the synchronous reactive modeling [29], two ways are distinguished for modeling multi-clock systems. The first model assumes that system holds a *reference abstract clock* according to which its components activation is characterizable. Such an abstract clock is a discrete set logical instants. We refer to this model as *synchronous multi-clock model*. The Lustre language [130], its Lucid Synchrone variant [50] and Esterel [206] embrace this vision. The other model, referred to as *polychronous model* considers no reference clock in a multi-clock system. A major advantage of the polychronous model is that different system components can evolve in a loosely-synchronous fashion, which is quite adapted for capturing GALS executions. It also favors composability by enabling modular or incremental design. The polychronous model is adopted by the Signal language [26], its MRICDF variant [144] and the CCSL language [178].

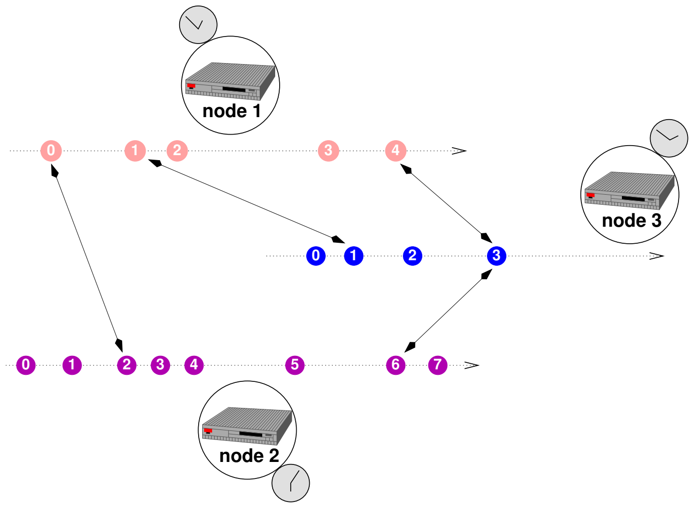


Figure 4: A multi-clocked GALS system.

An example of polychronous model of a GALS system is illustrated in Figure 4. Events are represented by bullets labeled with their occurrence rank according to their corresponding time scale (a horizontal line). The interactions between the three illustrated nodes can be represented using synchronization relations between event occurrences, *e.g.*, first event occurrence (tagged "0") of `node 1` and third event occurrence (tagged "2") of `node 2`, second event occurrence of `node 1` and second event occurrence of `node 3`, etc. From an overall viewpoint of a system, these relations only yield a partial occurrence ordering of all observed events; while focusing on a node, all its local events are totally ordered with respect its clock.

2.1.1 *Dealing with asynchrony with the polychronous model*

The design of distributed systems has been extensively studied for decades [226, 67]. The asynchrony [86] inherent to these systems appears *a priori* as an obstacle to their description with the synchronous model. According to [236], a fully synchronous system is characterized by the *boundedness and knowledge* of: *i) processing speed*, *ii) message delivery delay*, *iii) local clock rate drift*, *iv) load pattern*, and *v) difference among local clocks*. A fully asynchronous system assumes none of these characteristics.

The polychronous model aims at a modular design of systems with multiple loosely coupled clocks in order to deal with the complexity of their distribution. Compared to the synchrony/asynchrony definition of [236], it offers an intermediate vision: while it assumes the boundedness of computation and communication activities, the difference between local activation clocks of system parts is *a priori* unknown. So, my first contribution aims to answer the following question:

First challenge.

How to define a pragmatic approach for the correct design of distributed embedded systems with the polychronous model?

The definition of synchronous models of asynchronous mechanisms, e. g., for communication and synchronization, is one key ingredient to a successful solution to the above issue. Another important ingredient consists in exploiting the clock properties of polychronous models for their refinement towards desynchronized designs. Finally, bringing all these capabilities at a user-friendly level is very important for the pragmatic application of the advocated vision.

2.1.2 *Static analysis for polychronous designs*

Abstract clocks play a central role in the reasoning on polychronous designs. They denote sets of logical instants at which events occur. Typically, to prove the reactivity of a system with respect to an event, one can check whether or not the associated clock is empty. The mutual exclusion of two different events is checked by verifying that their clock intersection is empty. Beyond such properties, the automatic code generation from polychronous specifications uses abstract clocks to infer optimized control structures in resulting sequential or distributed code. This is also the case for synchronous multi-clock models in Lustre [37] for producing efficient sequential code.

In a polychronous language such as Signal, clock properties are analyzed statically during the compilation of specifications. The quality of the static analysis and the code generation performed by the compiler quite depend on the efficiency of the clock analysis. The clock analysis usually relies on a Boolean abstraction of programs, internally represented as *binary decision diagrams* (BDD) [45] for an efficient reasoning [10]. While both the status (i.e., presence/absence) and values of Boolean signals are fully taken into account, only the status of non Boolean signals is considered in this abstraction. As a result, when the clock properties expressed in a program are defined on numerical expressions, such an abstraction misses relevant information about their values. This sometimes leads to inaccurate analysis and inefficient code generation. Regarding this issue, my second contribution addresses the following question:

Second challenge.

How to improve the static analysis of polychronous specifications for a better clock analysis and automatic code generation?

The ability to efficiently address design properties requires an investigation of new abstractions for polychronous languages. In the case of Signal, the candidate abstractions must enable to tackle both logical and numerical properties.

2.2 POLYCHRONOUS DESIGN OF DISTRIBUTED EMBEDDED SYSTEMS

In the next sections, I start with an overview of related works on the synchronous design of distributed embedded systems. Then, I present my main contributions regarding this topic.

2.2.1 *Some related works in the synchronous approach*

There have been numerous studies on program distribution in synchronous languages. Most of these studies focus on *automatic program distribution* methods [111]: given a centralized synchronous program P and a distributed architecture A, the deployment of P on A is defined automatically, with the necessary inserted communication code, so that the resulting distributed program has the same functional behavior as P. Beyond these studies, automatic program distribution has been widely investigated [122].

The major contributions on this topic in the synchronous approach community could be summarized according to the main synchronous languages [29]: Esterel, Lustre and Signal. In Esterel, we mention the work of Berry and Sentovich [31] on the construction of GALS systems as synchronous circuits represented by a network of communicating *codesign finite state machines*. GALS architectures [53] consist of components that execute synchronously and communicate asynchronously. Another relevant work concerns the Esterel specification and programming of large-scale distributed real-time systems [125].

The Lustre language has been also used in several studies on the design of distributed systems. Girault [111] addressed the distribution of synchronous automata within the framework of this language. Afterward, he focused on further issues such as automatic deduction of GALS systems from centralized synchronous circuits, and the optimized execution of desynchronized embedded reactive programs to guarantee real-time constraints [113]. We must note the very important achievements by Caspi and his colleagues on the same topics [111]. They proposed an approach to deploy Lustre programs on *time-triggered architectures* [51]. The synchronous modeling of asynchronous mechanisms has been studied by Halbwachs in [126, 127]. Recently, with colleagues, he addressed the expression of complex scheduling policies managing shared resources [141]: priority inheritance protocol and priority ceiling protocols. This work served to translate AADL (Architecture Analysis & Design Language) specifications into Lustre, which enabled the application of model-checking to verify their properties.

In Signal, there have been also lots of results on program distribution. The earlier work of Chéron [56] dealt with the communication of separately compiled Signal programs. Then, Le Goff [158] proposed a decomposition of Signal programs into clustering models. Maffeïs [176] showed how to abstract such programs into graphs in order to define the qualitative scheduling and partitioning of these graphs. The work of Aubry [18] focused on similar problems as Girault by exploring the manual and semi-automatic distribution of synchronous dataflow programs in Signal. While these studies were mostly devoted to the practical side, Benveniste and Le Guernic lead several theoretical works on the distribution of Signal programs [25, 27, 108, 28, 163, 186].

Finally, we mention other interesting contributions such as [118] in which Grandpierre showed, with Petri nets, how one can derive a distributed implementation from a synchronous dataflow specification, e.g. in Lustre or Signal, of a given application while minimizing the response time w.r.t. real-time requirements. The SynDEx environment [119] aims at providing designers with such program distribution facilities.

2.2.2 *Modeling of asynchronous mechanisms in Signal*

Key publication for more details on this work: [99].

During my PhD thesis, I have defined in Signal a library of component models that form building blocks for the description of embedded real-time systems by taking into account their asynchronous features [99]. The definition of these components relies on the APEX-ARINC standard, which is dedicated to the design of integrated modular avionics architectures. This standard specifies two main multitasking levels for system execution by distinguishing the notions of *partition* and *process*. Partitions are logical allocation units resulting from a functional decomposition of a system. They are grouped into different modules. A processor is allocated to each partition for a fixed time window within a major time frame maintained by a *module-level OS*. Partitions communicate asynchronously via logical *ports* and *channels*.

Partitions are composed of *processes* representing the executive units that run concurrently. Each process is uniquely characterized by information like its period, priority or deadline, useful to the *partition-level OS* which is responsible for the correct execution of processes within a partition. The scheduling policy for processes is priority preemptive. Communications between processes are achieved by three basic mechanisms. The bounded *buffer* enables to send and receive messages following a FIFO policy. The *event* permits the application to notify processes of the occurrence of a condition for which they may be waiting. The *blackboard* is used to display and read messages: no message queues are allowed, and any message written on a blackboard remains there until the message is either cleared or overwritten by a new instance. Synchronizations are achieved using a *semaphore*.

Beyond communication, synchronization, partition/process management services, the APEX-ARINC standard defines further services for: and time and error management. The whole set of services I modeled in Signal is described in [94] [89]. Furthermore, a few communication component models have been defined in [90] [95]. The whole component models are made available as a library within the Polychrony design environment of Signal. They served in several studies that target Polychrony in order to benefit from its associated formal tools: analysis and optimization of real-time implementation of embedded software in Java [224], performance analysis [91], analysis and simulation of AADL models of safety-critical applications [175], model-driven engineering for embedded applications [44].

2.2.3 *A methodology for correct distributed design*

Key publication for more details: [95].

I have investigated a design methodology for distributed embedded systems on top of the previous component models [95]. The aim is to provide the adequate means to clearly express the inherent concurrency/parallelism of embedded applications and to validate the resulting behavior w.r.t. functional and non functional requirements.

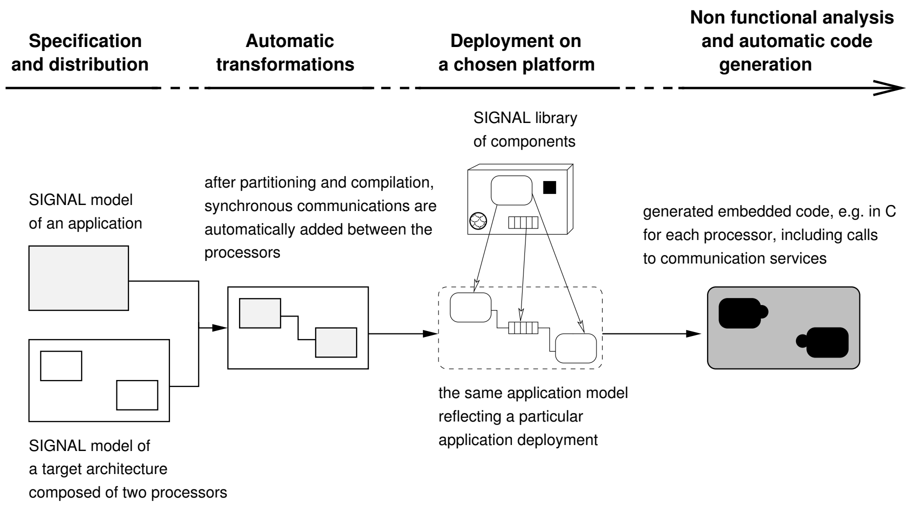

Specification and distribution
Automatic transformations
Deployment on a chosen platform
Non functional analysis and automatic code generation
SIGNAL model of an application
SIGNAL model of a target architecture composed of two processors
after partitioning and compilation, synchronous communications are automatically added between the processors
SIGNAL library of components
the same application model reflecting a particular application deployment
generated embedded code, e.g. in C for each processor, including calls to communication services

Figure 5: Overview of our design methodology for distributed embedded systems.

OVERVIEW. This methodology targets multi-processor architectures and fully benefits from the multi-clock property of the Signal language. It consists of four steps (see Figure 5) achieved in the Polychrony design environment as follows:

1. *System specification and manual distribution*: modeling of application functionality, modeling of distributed hardware architecture, and the mapping of the former on the latter. This mapping can be decided either directly from a given application model or after some application partitioning into clusters as shown in [99]. This step is usual in hardware/-software codesign approaches.

2. *Automatic transformations*: preserving the global functional correctness of the system after distribution. For this purpose, the compiler applies some transformations that exploit the endochrony and endo-isochrony properties of specifications [163, 27]. The endochrony property is the ability of a synchronous program P to execute in an environment that only provides P with the values of its inputs, without any information about their status (present or absent). In other words, P is able to

reconstruct a unique synchronous behavior from any external (asynchronous) input sequence of values. The endo-isochrony property provides sufficient conditions under which the synchronous composition of a pair of programs P_1 and P_2 is equivalent to their asynchronous composition (i.e. involving "*send/receive*"-like communications). This ensures that P_1 and P_2 can be safely deployed on GALS architectures.

3. *Deployment on specific platforms*: instantiating the different parts of the model resulting from the transformation of the compiler with components that represent specific platform mechanisms, e.g. for communication or synchronization. These components are taken from the library presented in the previous section.

4. *Analysis and automatic code generation*: checking the functional and non functional (temporal) constraints induced by the chosen deployment and generating a distributed code. This last step uses the static clock analysis provided by the Signal compiler together with non functional analysis techniques proposed in [219, 154] and implements code generation approach defined in [108].

The above steps are to be considered in an iterative design style. According to user-expected properties evaluated in the last step, the system can be redesigned in the first step again so as to go through the next steps until a satisfactory solution is obtained.

AN APPLICATION. In [95] with Thierry Gautier, we have showed how the above methodology is applied on a simple case study consisting of a Flight Warning System (FWS) from the avionic domain. This system is used in the Airbus A340 aircraft and has been proposed by the Aerospatiale Company (France) [172]. It is in charge of deciding on when and how to emit warning signals whenever there is an anomaly during the operational mode of an airplane. It is illustrated in Figure 6 together with its implementation architecture. It consists of two cyclic concurrent processes:

- given an alarm a_i, the *alarm manager* process confirms a_i after a given period of time or removes a_i from the set of confirmed alarms depending on the fact that a_i is detected "present" or "absent";
- the *alarm notifier* process emits warning signals associated with confirmed alarms.

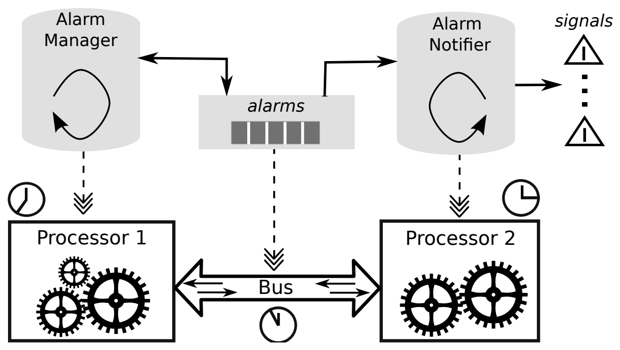


Figure 6: Deployment of FWS on a platform.

We used the affine clock relations defined in Signal [219] to address the synchronizability issues that can be raised in the resulting distributed multi-clock design.

2.2.4 *Model-driven engineering for polychronous design*

An important ingredient for the success of a design methodology is its companion user environment, which facilitates its applicability. Providing very simple and intuitive design concepts is of prime importance. For that purpose, with colleagues from the Espresso group, we have proposed the Signal-Meta environment (SME) [44] as an Eclipse plug-in. This work has been funded by the French RNTL project, named OpenEmbeDD[1]. Signal-Meta aims to serve as a pivot modeling tool for a customized computer-aided engineering of embedded systems starting from multiple and heterogeneous initial specifications. It is defined on top of Polychrony. Automated transformations are implemented in order to produce, analyze, statically verify and model-check Signal programs obtained from high-level models described in Signal-Meta.

Key publication for a detailed overview: [44].

In Signal-Meta, I was mainly involved in its integrated modular avionics design part [97], which has been defined during the Master internship of Romain Delamare during his Master internship in Espresso group in 2005, under my supervision.

In order to enable an easy definition of combined imperative and declarative specifications, we proposed an extension in Signal-Meta for control-oriented behaviors as in Mode Automata [179] and Lucid Synchrone [60] languages. The basic principle in these languages is to mix declarative dataflow statements with a notion of mode, representing the states (or configurations) according to which computations are achieved. In Signal-Meta, our proposed extension distinguishes itself from the other extensions in that it combines polychronous dataflow statements with the notion of modes. We refer to it as *polychronous mode automata* [225].

The definition of SME favored the integration of the Polychrony tool-set in the OpenEmBeDD model-driven engineering (MDE) open-source platform for the design real-time and embedded systems. The ultimate goal is to make the polychronous design paradigm and associated tools out of reach for a wide range of designer profiles.

2.3 STATIC ANALYSIS OF POLYCHRONOUS SPECIFICATIONS

A few works about the static analysis of synchronous programs are first surveyed. Then, our proposal for adopting satisfiability modulo theory (SMT) to deal with both logical and numerical properties is exposed.

2.3.1 *Some related works on synchronous languages*

The most relevant works on the static analysis of synchronous programming regarding combined logical and numerical properties have been achieved for the Lustre and Signal languages. Some of them are summarized below.

For several years, there have been significant efforts to combine numerical and Boolean techniques for the verification of Lustre programs. In [142], the technique used is a dynamic partitioning of the control flow obtained by

1 http://openembedd.org/home_html

Lustre compilation (which contains a few number of control points) with respect to constraints coming from a given proof goal.

In [124], SMT is used to verify safety properties on Lustre programs. The authors consider a specific form of Lustre language and propose a modeling in a typed first order logic with uninterpreted function symbols and built-in integers and rationals. While this work also aims at benefiting from SMT solving in synchronous programming, it misses all useful clock analysis achieved by the Signal compiler. Such an analysis includes suitable heuristics to address multi-clocked specifications. Neither an SMT solver nor the Lustre compiler makes this analysis possible.

An important work is the polyhedral-based static analysis for synchronous languages of [34]. The authors give a technique based on fix-point iteration on a lattice combining Boolean and affine constraints. The technique we advocate for Signal is less precise because it only uses interval approximation. However, the complexity in our case is lesser and the implementation is much simpler.

An inspiring relevant study presented in [188], concerns the definition of a clock language $\mathcal{CL}$ aiming to capture the static control part of Signal programs. The author also considers SAT decision procedures to prove clock properties. However, statements involving the *delay* construct are not taken into account in this study. This reduces the scope of the proposed analysis. Our proposition covers all Signal programs and offers more expressivity than $\mathcal{CL}$.

Finally, In [98, 100], we already proposed a preliminary solution to the static analysis of combined logical/numerical properties of Signal programs. An interval-based data structure referred to as *interval-decision diagram* (IDD) is considered a package for the analysis of numerical properties in Signal programs. I have re-implemented this package together with Gilles Atigossou during his Master internship in DaRT group of LIFL and Inria (Lille) under my supervision, from February 2008 to June 2008. I will discuss the limitation of this initial solution later.

2.3.2 *A new abstraction for Signal*

I give an overview of my recent proposal, in collaboration with Laure Gonnord, for improving the static analysis of combined logical/numerical properties of Signal programs. This proposal relies on an abstraction of language statements. I first recall them before presenting the abstraction. Afterward, an example is given for illustration.

Key publication for a complete presentation: [96].

A SUMMARY OF SIGNAL CONSTRUCTS. Signal handles unbounded series of typed values $(x_t)_{t \in \mathbb{N}}$, called *signals*, implicitly indexed by discrete time, and denoted as x. For instance, a signal can be either of *Boolean* or *integer* or *real* types. At any logical instant $t \in \mathbb{N}$, a signal may be present, at which point it holds a value; or absent and denoted by $\perp$ in the semantic notation. There is a particular type of signal called `event`. A signal of this type always holds the value *true* when it is present. The set of instants at which a signal x is present is referred to as its *clock*, noted ^x. A *process* is a system of equations over signals, specifying relations between values and clocks of the signals. A *program* is a process. Signal relies on six primitive constructs defining the *core language* as follows:

Instantaneous relations: `y:= R(x1,...,xn)` where `y, x1`, ..., `xn` are signals and `R` is a point-wise n-ary relation/function extended canonically to signals. This construct imposes `y, x1`, ..., `xn` *i)* to be simultaneously present, i.e. `^y = ^x1 = ...= ^xn` (i.e. *synchronous* signals), and *ii)* to hold values satisfying `y:= R(x1,...,xn)` whenever they occur.

Delay: `y:= x $ 1 init c` where `y, x` are signals and `c` is an initialization constant. It imposes *i)* `x` and `y` to be synchronous, i.e. `^y = ^x`, while *ii)* `y` must hold the value carried by `x` on its previous occurrence.

Under-sampling: `y:= x when b` where `y, x` are signals and `b` is of Boolean type. This construct imposes *i)* `y` to be present only when `x` is present and `b` holds the value *true*, i.e. `^y = ^x` $\cap$ $[b]$ (where $[b] \cup [\neg b] =$ `^b` and $[b] \cap [\neg b] = \emptyset$), while *ii)* `y` holds the value of `x` at those logical instants. Note that the sub-clock $[b]$ (resp. $[\neg b]$) denotes the set of instants where `b` is *true* (resp. *false*).

Deterministic merging: `z:= x default y` where `z, y, x` are signals. This construct imposes *i)* `z` to be present when either `x` or `y` are present, i.e. `^z = ^x` $\cup$ `^y`, while *ii)* `z` holds the value of `x` uppermost, otherwise that of `y`.

Composition: $P \equiv P_1 | P_2$ where P_1 and P_2 are processes. It denotes the union of equations defined in processes, leading to the conjunction of the constraints associated with these processes. The composition operator is commutative and associative.

Restriction (or Hiding): $P \equiv P_1$ `where x`, where P_1 and `x` are a process and a signal. It enables local declarations in process P_1, and leads to the same constraints as P_1.

A BOOLEAN-INTERVAL ABSTRACTION. To define an abstraction for Signal program analysis, all considered programs are supposed to be in the syntax of the core language. The abstract semantics of a program, is characterized by a set of valuation couples of the form $(\widehat{\ }, \widetilde{\ })$, defined by the following functions:

- $\widehat{\ }: X_P \to \mathbb{B} = \{true, false\}$ assigns to a variable a Boolean value;
- $\widetilde{\ }: X_P \to \mathbb{R} \cup \mathbb{B}$ assigns to a variable a numerical or Boolean value.

Intuitively, in a couple $(\widehat{x_i}, \widetilde{x_i})$, the first component of a valuation couple encodes the clock of a signal, where $true$ and $false$ respectively mean presence and absence of instant in the clock. The second component encodes the value taken by a signal according to its presence, i. e., when $\widehat{x_i}$ is $true$.

The set of all possible valuation couples associated with a Signal process or program can be represented as a first order logic formula Φ_P in which atoms are $\widetilde{x_i}$ and $\widehat{x_i}$, and the operators are usual logic operators and integer comparison functions.

We define the abstraction of programs by considering only a subset of numerical and Boolean expressions in statements of Signal. The abstraction of these expressions is defined by induction on their structure as shown in [96]. Here, I will only show the abstraction Φ of core statements. This is summarized in Figure 7. Two possible definitions of Φ are distinguished according to the type of defined signal `y` in each equation: (1) when `y` is of numerical type and (2) when `y` is of logical type.

Our abstraction is sound, in the sense that it preserves the behaviors of the abstracted programs. In other words, if a property is true on the abstraction, then it is also the case on the program:

Primitive statements P	Corresponding abstractions (logic formulas Φ_P)	Comments
`y:= R(x1,...,xn)`	$\begin{cases} \bigwedge_{i=1}^{n}(\widehat{y} \Leftrightarrow \widehat{x_i}) \wedge \left(\widehat{y} \Rightarrow \widetilde{y} \in \phi(nexp)\right) & (1) \\ \bigwedge_{i=1}^{n}(\widehat{y} \Leftrightarrow \widehat{x_i}) \wedge \left(\widehat{y} \Rightarrow (\widetilde{y} \Leftrightarrow \phi(bexp))\right) & (2) \end{cases}$	Signal variables `y` and `xi` are abstracted by the couples $(\widehat{y}, \widetilde{y})$ and $(\widehat{x_i}, \widetilde{x_i})$ respectively denoting the clock and value encodings of `y` and `xi`; the expression `R(x1,...xn)` is abstracted by $nexp$ or $bexp$ depending on whether it is of numerical type or Boolean type.
`y:= x$1 init c`	$\begin{cases} (\widehat{y} \Leftrightarrow \widehat{x}) \bigwedge (\widehat{y} \Rightarrow (\widetilde{y} = \widetilde{x} \vee \widetilde{y} = c)) & (1) \\ (\widehat{y} \Leftrightarrow \widehat{x}) \bigwedge (\widehat{y} \Rightarrow (\widetilde{y} \Leftrightarrow (\widetilde{c} \vee \widetilde{x}))) & (2) \end{cases}$	When `y` is of numerical type, a classical interval analysis would perform the convex union of the two intervals $\widetilde{c}$ and $\widetilde{x}$. Here, the approximation resulting from such a convex union is avoided by keeping the disjunction as is.
`y:= x when b`	$\begin{cases} (\widehat{y} \Leftrightarrow (\widehat{x} \wedge \widehat{b} \wedge \widetilde{b})) \bigwedge (\widehat{y} \Rightarrow \widetilde{y} = \widetilde{x}) & (1) \\ (\widehat{y} \Leftrightarrow (\widehat{x} \wedge \widehat{b} \wedge \widetilde{b})) \bigwedge (\widehat{y} \Rightarrow (\widetilde{y} \Leftrightarrow \widetilde{x})) & (2) \end{cases}$	
`y:= x default z`	$\begin{cases} (\widehat{y} \Leftrightarrow (\widehat{x} \vee \widehat{z})) \bigwedge \left(\widehat{y} \Rightarrow ((\widehat{x} \wedge (\widetilde{y} = \widetilde{x})) \vee (\neg\widehat{x} \wedge (\widetilde{y} = \widetilde{z})))\right) & (1) \\ (\widehat{y} \Leftrightarrow (\widehat{x} \vee \widehat{z})) \bigwedge \left(\widehat{y} \Rightarrow ((\widehat{x} \wedge (\widetilde{y} \Leftrightarrow \widetilde{x})) \vee (\neg\widehat{x} \wedge (\widetilde{y} \Leftrightarrow \widetilde{z})))\right) & (2) \end{cases}$	
$P_1 \mid P_2$	$\Phi_{P1} \wedge \Phi_{P2}$	
`P where x`	$\exists\widetilde{x}, \exists\widehat{x} \,.\, \Phi_P$	

Figure 7: Summary of Boolean-Interval abstraction of Signal.

Proposition 1 *Given a program P and a formula φ in which atoms are $\hat{x_i}$ and $\widetilde{x_i}$ ($x_i \in X_P$), if $\Phi_P \Rightarrow \varphi$, then $[\![P]\!] \subseteq \Gamma(\varphi)$. P is said to satisfy φ.* □

The complete proof of this proposition is given in [96]. To check $\Phi_P \Rightarrow \varphi$, we use Satisfiability Modulo Theory (SMT), which checks the satisfiability of formulas over multiple theories such as Boolean, Integer, etc. [69]. We mainly consider the Yices SMT solver version in our implementation.

COMPARISON WITH OUR PRELIMINARY SOLUTION. In [98, 100], we already suggested a preliminary solution for the static analysis based on IDDs. The choice of SMT solvers in the new solution appears more judicious. First, in IDDs, intervals are only defined on integers. As a result, to deal with other numerical types such as reals, IDDs require a prior encoding into integers. With SMT solvers, a wide range of arithmetic theories are made possible, which allows a more expressive analysis without much effort compared to IDDs. Second, from a practical point of view, the integration of IDDs in the Signal compiler is more difficult since it requires a very careful coupling with the other data structures used during the static analysis. One important question is how to make efficient and costless the management of binary decision diagrams (BDDs), which are part of IDDs and are already present in the compiler.

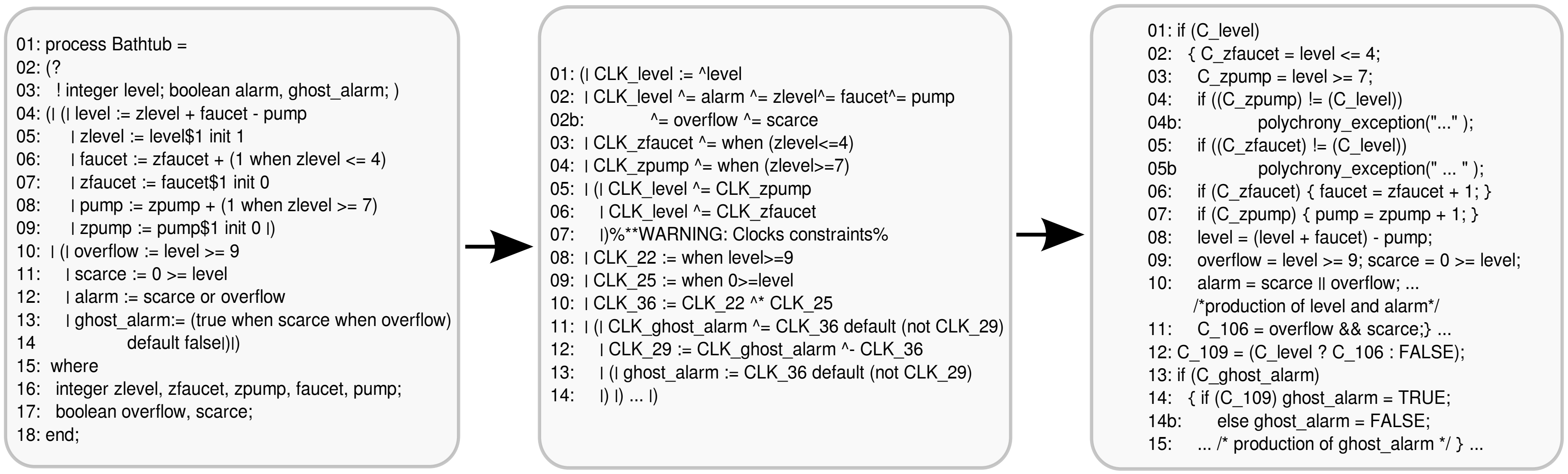
In the new solution, we rather consider a *non intrusive* solution regarding the compiler, which consists in deducing additional information from an initial program specification with SMT solvers. This therefore enables the compiler to have an explicit and rich set of constraints for a better program analysis and code analysis by using its current clock calculus technique.

On the other hand, compared to [142], our approach is not dependent on any proof goal, and the Boolean variables are not hidden in the control (except for the step 1). In addition, Lustre compilation [132] suffers from the same lack of precision concerning numerical variables. Indeed, no numerical analysis is done during compilation. Hence, our method could be considered for improvement.

2.3.3 *Application to Signal clock calculus: an example*

The Signal process shown in Figure 8(a) specifies the status of a *bathtub* [34]. It has no input signal (line 02), but has three output signals (line 03). The signal `level`, defined at line 04, reflects the water level in the bathtub at any instant. It is determined by considering two signals, `faucet` and `pump`, which are respectively used to increase and decrease the water level. These signals are increased by one under some specific conditions (lines 06 and 08), in order to maintain the water level in a suitable range of values.

An `alarm` signal is defined at line 12 whenever the water overflows (line 10) or becomes scarce (line 11) in the bathtub. An additional "ghost" alarm is defined at line 13/14, which is not expected to occur. Here, it is just introduced to illustrate one limitation of the static analysis of Signal. The clock of this signal is not completely specified in `Bathtub`. As stated in the previous section, this clock is the union of those associated with the two arguments of the `default` operator. The clock of the left argument is exactly known. The clock of the right-hand one is *context-dependent because the argument is a constant*: it is equal to the difference of `ghost_alarm`'s clock and first argument's clock. Since, this difference cannot be defined exactly from the program, further clock constraints on `ghost_alarm` will be required from the environment of `Bathtub` for an execution.



```
01: process Bathtub =
02: (?
03:   ! integer level; boolean alarm, ghost_alarm; )
04: (| (| level := zlevel + faucet - pump
05:      | zlevel := level$1 init 1
06:      | faucet := zfaucet + (1 when zlevel <= 4)
07:      | zfaucet := faucet$1 init 0
08:      | pump := zpump + (1 when zlevel >= 7)
09:      | zpump := pump$1 init 0 |)
10:  | (| overflow := level >= 9
11:     | scarce := 0 >= level
12:     | alarm := scarce or overflow
13:     | ghost_alarm:= (true when scarce when overflow)
14                  default false|)|)
15:  where
16:   integer zlevel, zfaucet, zpump, faucet, pump;
17:   boolean overflow, scarce;
18: end;
```

(a)

Bathtub specification in Signal

```
01: (| CLK_level := ^level
02:  | CLK_level ^= alarm ^= zlevel^= faucet^= pump
02b:             ^= overflow ^= scarce
03:  | CLK_zfaucet ^= when (zlevel<=4)
04:  | CLK_zpump ^= when (zlevel>=7)
05:  | (| CLK_level ^= CLK_zpump
06:     | CLK_level ^= CLK_zfaucet
07:     |)%**WARNING: Clocks constraints%
08:  | CLK_22 := when level>=9
09:  | CLK_25 := when 0>=level
10:  | CLK_36 := CLK_22 ^* CLK_25
11:  | (| CLK_ghost_alarm ^= CLK_36 default (not CLK_29)
12:     | CLK_29 := CLK_ghost_alarm ^- CLK_36
13:     | (| ghost_alarm := CLK_36 default (not CLK_29)
14:     |) |) ... |)
```

(b)

Sketch of the clock calculus result

```
01: if (C_level)
02:   { C_zfaucet = level <= 4;
03:     C_zpump = level >= 7;
04:     if ((C_zpump) != (C_level))
04b:              polychrony_exception("..." );
05:     if ((C_zfaucet) != (C_level))
05b               polychrony_exception(" ... " );
06:     if (C_zfaucet) { faucet = zfaucet + 1; }
07:     if (C_zpump) { pump = zpump + 1; }
08:     level = (level + faucet) - pump;
09:     overflow = level >= 9; scarce = 0 >= level;
10:     alarm = scarce || overflow; ...
       /*production of level and alarm*/
11:     C_106 = overflow && scarce;} ...
12: C_109 = (C_level ? C_106 : FALSE);
13: if (C_ghost_alarm)
14:   { if (C_109) ghost_alarm = TRUE;
14b:       else ghost_alarm = FALSE;
15:     ... /* production of ghost_alarm */ } ...
```

(c)

Sketch of automatically generated C code

Figure 8: Static analysis and code generation for a bathtub model in Polychrony.

ANALYSIS BASED ON USUAL BOOLEAN ABSTRACTION. Figure 8(b) partially shows the result of the clock calculus generated automatically by the Signal compiler, based on the Boolean abstraction. Here, we focus on two issues that the clock analysis was not able to fix adequately:

- Lines `05–07`: a clock constraint is generated, stating signals `CLK_level`, `CLK_zfaucet` and `CLK_zpump` must have the same clock, while signals `CLK_zfaucet` and `CLK_zpump` have exclusive clocks (lines `03–04`);
- Line `11`: the right-hand side of the synchronization equation defining the signal `CLK_ghost_alarm` should be `(not CLK_29)` since the clock `CLK_36` is empty by definition (line `10`).

These issues illustrate the limitations of the Boolean abstraction for clock properties, e.g., clock exclusion or emptiness, involving numerical expressions. A more expressive clock analysis would detect the fact that `CLK_level`, `CLK_zfaucet` and `CLK_zpump` must be empty clocks in order to satisfy the clock constraints of the `Bathtub` process. In addition, these limitations have an important impact on the quality of the code generated automatically by the compiler since it relies on the clock hierarchy resulting from the analysis. Figure 8(c) sketches a C code generated automatically based on the clock analysis. The previous clock constraint is implemented by exception statements (lines `04–05`). Since `CLK_level`, `CLK_zfaucet` and `CLK_zpump` should be empty clocks, statements between lines `02` and `11` are never executed, i.e. a *dead code*. Similarly, the `if` statement at line `14/14b` also contains a dead code since the variable `ghost_alarm` is always set to *false*. Efficient dead code elimination is of high importance in compilers[68].

ANALYSIS BASED ON THE NEW ABSTRACTION. We illustrate the use of the previous analysis in the analysis of the `Bathtub` (see Figure 8(a)). First, the abstraction $\Phi_{\texttt{Bathtub}}$ of the `Bathtub` program is computed. Then, we have to specify the clock synchronization and clock emptiness properties (formula φ) of interest for the program. Among these properties, let us focus on the following:

- `pump` and `faucet` have disjoint clocks: $\neg(\widehat{faucet} \wedge \widehat{pump})$
- `overflow` and `scarce` cannot be true at the same time: $\neg(\widetilde{scarce} \wedge \widetilde{overflow} \wedge \widehat{scarce} \wedge \widehat{overflow})$
- `alarm` and `level` have the same clock: $\widehat{alarm} \Leftrightarrow \widehat{level}$

We consider the formula $\Phi_{\texttt{Bathtub}} \wedge \neg\varphi$, where φ denotes the property to be checked. With the Yices solver, we get for all properties *unsat*, which means that $\Phi_{\texttt{Bathtub}} \models \varphi$. Thanks to Proposition 1, the property φ is satisfied by `Bathtub`. Here, the previous three formulas are proven. Then, the program `Bathtub` is composed with the following Signal statements that concretize these properties:

- `faucet ^* pump ^= ^0`
- `true when scarce when overflow ^= ^0`
- `alarm ^= level`

The result obtained from the analysis of the new process by the compiler is illustrated in Figure 9 shows that the compiler was able to infer that the `ghost_alarm` signal is always equal to `false` (line `01`). The compiler now detects that the clocks of all other signals are empty (line `04/04b`).

```
---------------------------------------------------
(| CLK_ghost_alarm := ^ghost_alarm
 | CLK_ghost_alarm ^= ghost_alarm
 | (| ghost_alarm := not CLK_ghost_alarm |)
 |);%^0 ^= level ^= alarm
        ^= zlevel ^= zfaucet ^= zpump
      ***WARNING: null clock signals%
---------------------------------------------------
```

Figure 9: A sketch of the clock calculus for `Bathtub_Bis`.

Beyond, the detection of empty clock presence in the `Bathtub_Bis` program, the automatically generated C code also benefits from the new clock analysis performed by the compiler. As a result, the corresponding generated code provided in Figure 10, is now optimized in the sense that no useless code fragment appears.

```
--------------------------------------------------
  { ghost_alarm = FALSE;
   /* produce output value
        for the signal ghost_alarm */ } ...
--------------------------------------------------
```

Figure 10: A sketch of the C code for `Bathtub_Ter`.

2.3.4 *Static analysis of polychronous programs in MRICDF*

Key publications for more details: [146] and [147].

In April 2010, I have been invited by Sandeep Shukla in his lab at Virginia Tech (VA, USA) for a short stay in order to discuss about the static analysis of the Multi-Rate Instantaneous Channel connected Data Flow (MRICDF) language. Sandeep and his students have been working of this variant language of Signal for the development of safety-critical applications. I already had some collaborations with Sandeep's group during my PhD thesis in the context of a project funded by NSF and Inria.

MRICDF [144] is a visual actor-oriented polychronous formalism, strongly inspired by Signal. Its primitive actors correspond to the Signal primitives statements on signals. It manipulates signals and refers to an abstract clock as an *epoch*. The static analysis of MRICDF specifications and associated code generation rely on epoch analysis by considering a Boolean encoding. The resulting system of Boolean equations defines a theory which has all satisfying assignments for an encoded actor network. An *implicant* of this theory is a disjunctive clause that implies a Boolean formula. There can be several implicants for a formula. A *prime implicant* is a disjunctive clause that is not covered by any other implicant. When a prime implicant is a single positive Boolean literal b_x associated with a signal x, then b_x is true for any arbitrary instant of a network. Then, x is a *master trigger* signal. By iterating the prime implicant identification on the rest of disjunctive clauses, signals having lower epochs than already found prime implicants are determined. During this iteration, such signals are organized within the *follower set*, which gives a global activation order of signals. Such a set defines a unique program execution order. A program is said to be *synthesizable* if has a master trigger, a follower set and does not contain any causality cycle.

SMT-BASED SOLUTION FOR EFFICIENT STATIC ANALYSIS. In practice, the construction of follower sets requires computing all possible prime implicants since the used generator cannot identify a master trigger signal. This resulted in high synthesis time that must be reduced [145], especially for larger MRICDF examples. Beyond this issue, MRICDF is also concerned with the same limitations as Signal regarding numerical clock properties due to the adopted Boolean encoding.

Table 2: Time required to find master trigger signal

MRICDF	Number of	Time in seconds		
networks	actors	Initial	AET	SMT
Height Supervisor	5	0.35	0.37	0.094
Absolute	8	0.91	0.91	0.078
Factorial	8	0.33	0.33	0.09
Resettable Counter	8	0.927	0.562	0.099
Watchdog Timer	14	16.3	5.3	0.11
Producer-Consumer	15	21.4	16.3	0.12
Flight Warning System	17	8.1	1.03	0.1
GCD	19	1.35	0.89	0.13
pEHBH	14	0.79	0.75	0.125

We advocated SMT solvers as a solution by formulating the master trigger existence as a satisfiability problem [146]. The master trigger test is achieved by encoding the clock of a candidate signal and setting it to *false*. An SMT formula is constructed by composing this information with the encoding of a given MRICDF model. If this signal is actually a master trigger, the evaluation of the formula must indicate that it is unsatisfiable. After the determination of the master trigger, the follower set is built by iterating the master trigger identification steps. Table 2 compares the master trigger identification time for different MRICDF networks according to: the *initial* implementation of the Boolean theory approach with a prime implicant generator; AET, the first improvement based on actor network reduction [145]; and SMT solution proposed in [146], which outperforms enhancements by several orders of magnitude. As a global observation, using SMT is beneficial in improving the causality analysis of polychronous programs for efficient software synthesis.

2.4 PEDAGOGICAL IMPLICATION: A BOOK ON SIGNAL PROGRAMMING

After several years of practice in polychronous design using the Signal language, I have reported all the gained insights in a book [92], published by Springer by the end of 2009. As I explained in its preface, this book was the first attempt to provide a large public (scientists, practitioners and students) with a pedagogical presentation of the necessary rudiments for a successful and pragmatic usage of Signal programming. Writing it was the best way to share my proper experience about the extensive usage of Signal for the design of multi-clock embedded systems.

It is worth noting that since 2004, I have been giving a few lectures per year to Master level students on Signal programming. These lectures were given successively at University of Rennes 1 and University of Lille 1. This rich and exciting teaching experience strongly served me in writing the aforementioned book.

2.5 SUMMARY AND DISCUSSION

In this chapter, I presented a summary of my contributions on the polychronous design and analysis of distributed embedded systems specified in Signal. The aim is to promote a rapid virtual prototyping of embedded systems implemented on GALS architectures, according to the formal approach promoted by the synchronous approach.

ON POLYCHRONOUS DESIGN OF DISTRIBUTED EMBEDDED SYSTEMS. I first addressed the definition of an adequate design methodology and its applicability in the Signal design environment Polychrony. My proposition relies on a set of components defined in Signal to model various asynchronous mechanisms for communication, synchronization and execution. The usage of these components has been combined with specific program transformations to obtain a correct distributed design of an application. Furthermore, I advocated a model-driven engineering for polychronous design in order to make easier the access to this methodology and the facilities of Polychrony. Nevertheless, some additional efforts are required to fully automate the proposed methodology and to experiment it on real-life large scale systems beyond the presented proof-of-concept.

In my opinion, an important research direction in the future is the effective exploitation of polychronous modeling for MPSoCs, which are also distributed embedded systems. Thanks to its features, this modeling paradigm provides a useful abstraction to safely deal with concurrency in software to execute on both heterogeneous and homogeneous parallel platforms. Concurrency plays an important role in optimizing system performances.

On the other hand, most of achievements in polychronous and synchronous approaches do not fully take into account the quantitative aspects of MPSoC requirements such as performances (execution time and energy consumption) and other physical constraints (memory and bandwidth issues), induced by the deployment of systems on real-life platforms. Although the synchrony hypothesis is an abstraction that may be seen as a barrier to tackle these requirements, there has been a series of recent works that tend to relax it for a refined reasoning, e.g., N-synchrony [59], Signal affine clocks [219], scheduling of synchronous specifications in SynDEx [207], the Kiel Esterel processor [170]. I believe there is a real opportunity to widely bring the correct-by-construction design promoted by polychronous and synchronous models to the MPSoC design community while taking into account its needs. In the next chapters, I will show some efforts in this connection.

Another relevant perspective concerns the use of polychronous specifications as an internal reasoning model for higher-level system architecture description languages such as the Marte standard profile [194] or the architecture analysis and design language[2] (AADL). The Espresso group already started some investigations recently in this direction.

STATIC ANALYSIS OF POLYCHRONOUS SPECIFICATIONS. An important advantage of polychronous and synchronous design approaches is their ability to enable automatic synthesis of correct and efficient software implementation. Compilers play an important role for this purpose. The relevance of

2 http://www.aadl.info

the static analysis on which they rely has a direct impact on the quality of generated code. In this chapter, I also dealt with the static analysis of combined logical/numerical clock properties in polychronous specifications. We have studied a complementary way to the current Boolean abstraction used in compilers of Signal and MRICDF for the clock analysis. In order to improve the quality of clock property verification and the optimization of the generated code, a new abstraction encoded in satisfiability modulo theory (SMT) has been proposed for the pragmatic reasoning. This solution has been validated only in an *ad hoc* way. A tool suite is under construction to support the connection of SMT solvers to the Polychrony environment.

Among the perspectives to this contribution, it is worth mentioning a possible extension of the solution to tackle dynamic property analysis. Indeed, the current proposal considers a static approximation of programs, in which the dynamic variation of the values held by state signal variables (i. e.. those defined with the delay primitive statement) is not taken into account. In this aim, a combination of techniques such as model-checking with clock calculus could be investigated in compilation.

FINAL OPINION ON THE PRESENTED CONTRIBUTIONS. Within the synchronous programming community, the results presented in this chapter mostly contribute to the problematic regarding synchronous reactive modeling of (bounded) asynchrony. More generally, beyond this community, they also contribute to a wider understanding of synchronous programming in Signal language. The implementation of parts of them in the Polychrony environment and its Signal-Meta extension has been concretely serving in various collaborations of the Espresso group as well as a recent PhD thesis.

Beyond these observations, I think additional efforts still remain to do in the perspective of effectively bringing the design principles of the polychronous model to GALS designers. This necessarily includes further improvements of the existing methodology with companion design assistance tools.

Executive summary

MAIN COLLABORATIONS

- Espresso group (IRISA/Inria, Rennes)
- Fermat lab (Virginia Tech, VA – USA)

PROJECT

- French OpenEmbeDD RNTL project (partners: Airbus, Anyware Technologies, CEA, CS SI, France Telecom, Inria (Aoste, DaRT & Espresso groups), LAAS lab, Thales Aerospace, Thales R&D, Verimag lab, 2006 – 2009)

ADVISORY

- Romain Delamare (Master internship for three months in 2005, 100%)
- Gilles Atigossou (Master internship from February 2008 to August 2008, 100%)

SELECTED PUBLICATIONS

- *Conference*: ACM SIGPLAN/SIGBED conference on Languages, compilers, and tools for embedded systems (LCTES), 2011 [96]
- *Conference*: ACM/IEEE Ninth International Conference on Formal Methods and Models for Codesign (MEMOCODE), 2011 [146]
- *Book*: Springer book (260 pages), 2010 [92]
- *Journal*: IEEE Transactions on Parallel and Distributed Systems (TPDS), 2010 [95]
- *Journal*: Journal of Logic and Algebraic Programming (JLAP), 2009 [44]

CONTRIBUTION TO SOFTWARE:

- Polychrony and Signal-Meta (`http://www.irisa.fr/espresso/Polychrony`)

3 DESIGN MODEL FOR REACTIVE DATA-INTENSIVE APPLICATIONS

Contents

This chapter presents my contributions since my Post-doc in September 2005 in the DaRT group of LIFL, on the design and analysis of reactive data-intensive applications (see Figure 11). The presented works describe and reason on reactive behaviors via the specification of running modes and instants at which given actions take place in data-intensive applications. The so-called repetitive structure modeling (RSM) is considered for specifying *regular* data-intensive applications. The central question of combining the synchronous reactive approach and RSM to model reactive behaviors in data-intensive applications is definitely deserving of attention in our approach. Some of these works have been achieved in collaboration with colleagues from the Sardes group of Inria and LIG (Grenoble) and the Aoste group of Inria and I3S (Sophia Antipolis). They also cover the PhD thesis of Huafeng Yu started in October 2005 and defended in December 2008 (co-advised with Éric Rutten and Jean-Luc Dekeyser), and the short Post-doc fellowship of Mohamed Fellahi (October 2010 – February 2011).

The remainder of the chapter is organized as follows: in Section 3.1, I introduce the main challenges of interest; in Section 3.2, I present the necessary background for the exposed ideas by giving a rapid survey of parallel programming models followed by RSM; in Section 3.3 is exposed the first part of my contributions about the extension of RSM for supporting both static and dynamic aspects of computations; in Section 3.3, I present the rest of my contributions devoted to the use of the synchronous reactive approach to analyze RSM-based system designs; finally, in Section 3.5, I discuss the strengths, limitations and future directions to the presented works. An executive summary is also given, regarding the key points in my contributions highlighted in this chapter.

3.1 OVERVIEW OF MAIN CHALLENGES

3.1.1 *Reactivity in massively parallel computations*

Reactive systems [205] have been studied for several decades. They are characterized by a continuous interaction with their environment. The rhythm at which their reactions, i.e. receipt of inputs and computation of outputs,

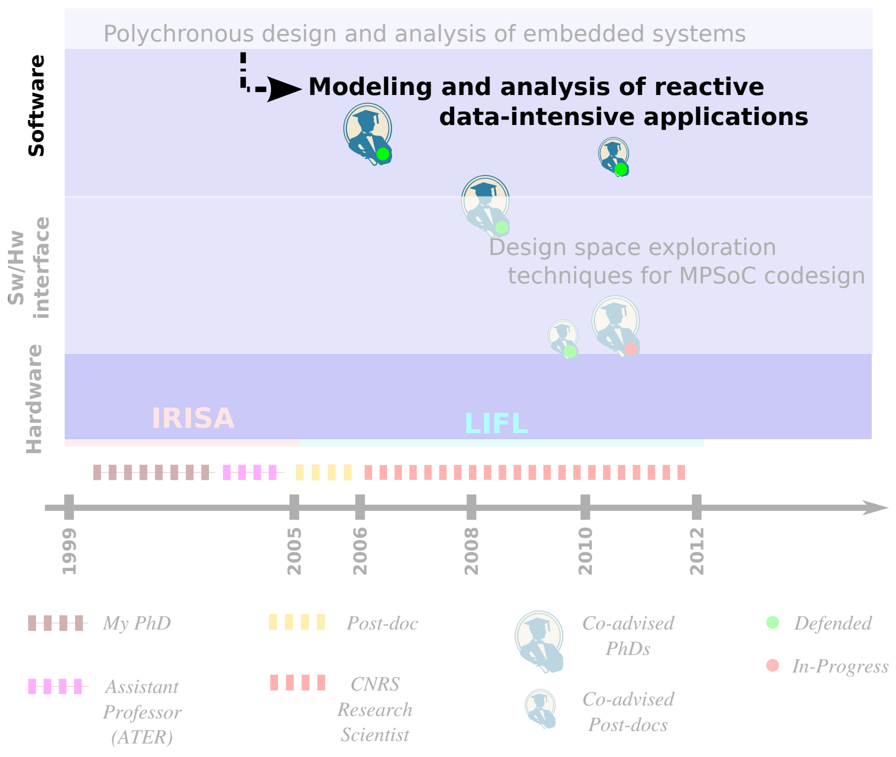

Polychronous design and analysis of embedded systems
Software
Modeling and analysis of reactive data-intensive applications
Sw/Hw interface
Design space exploration techniques for MPSoC codesign
Hardware
IRISA
LIFL
1999
2005
2006
2008
2010
2012
My PhD
Post-doc
Co-advised PhDs
Defended
In-Progress
Assistant Professor (ATER)
CNRS Research Scientist
Co-advised Post-docs

Figure 11: Specific contributions presented in the current chapter (the other contributions not exposed here are intentionally blurred).

is entirely under the control of the environment. In existing literature, there are a few studies devoted to the design of reactive data-intensive applications [220] [219]. The concerned applications include multimedia applications, which manipulate streams in the form of periodic signals. The events related to these signals define component activation and are synchronized when components interact. In the synchronous approach, this is well formalized with abstract clocks.

The *repetitive structure modeling* (RSM) [115] is a high-level specification formalism that enables to describe the potential parallelism in massively parallel systems in a very compact way. It uses repetitive data dependencies to express *regular and static* multidimensional data structures and system topologies. Thanks to these features, RSM offers a powerful expressivity for MPSoC system design. However, in order to describe reactive data-intensive applications on MPSoCs, we also need to capture the dynamic aspect of their behaviors: input data availability over the time (e.g., their rates), the effective exploitation of parallelism during executions according to resources and environment, etc. The first contribution in this chapter is devoted to the following question:

First challenge.

How to combine mode-oriented control and logical time with the repetitive structure modeling for the design of reactive data-intensive applications?

To answer the above question, we will borrow ideas from the synchronous reactive approach, by considering its abstract clock notion and mode-oriented design concepts to describe data rates and execution scenarios in reactive data-intensive applications. Loop transformations can be applied to RSM specifications for their refactoring so as to distinguish reactions in a consitent way.

3.1.2 *Design correctness of data-intensive applications*

Correct application specifications in RSM must satisfy basic properties of dataflow models such as single assignment, absence of causality cycles, etc. As shown in [41], checking such properties in RSM can be done by considering, e.g., Feautrier's work [83] [84] on techniques based on polyhedra and linear programming. Algorithms are proposed for dataflow analysis of array and scalar references. They mainly deal with pure data dependencies. While these solutions are very attractive, they are not necessarily well-adapted when data dependencies are combined with complex control flow.

Synchronous languages offer useful representations in which both data dependencies and control flow are analyzable uniformly. In Signal, such a representation is the *hierarchized conditional dependency graph* (HCDG) [163]. It combines data dependencies and activation clocks indicating when edges and vertexes of a dependency graph are valid w.r.t. control flow. The analysis of causality, determinism and single assignment also relies on this central structure. Moreover, other tools such as model-checkers connected to synchronous languages favor more sophisticated verification techniques. For all these reasons, I consider the synchronous technology to tackle the following issue:

Second challenge.

How to verify the correctness of reactive data-intensive application specifications in RSM with the synchronous approach?

To define an answer to this question, I will consider a translation of RSM specifications into synchronous dataflow programs on which adequate verification and analysis tools can be applied in order to deal with relevant properties of RSM. Among these tools, are considered compilers and model-checkers.

3.2 BACKGROUND NOTIONS

I give a rapid survey of parallel programming models, which are necessary ingredients for the design of data-intensive applications. This panorama is followed by the presentation of the basic features of RSM, which is considered for design in our contributions.

3.2.1 *A survey of parallel programming models*

A *programming model*[1] is characterized by a set of languages and libraries defining programmers view of a machine that executes an application [152]. It determines how a program is expressed. Parallel programming models used to combine two parts: a model dedicated to control, which describes how parallelism is managed; and another model dedicated to communication, which allows for the interaction between parallel entities by exchanging data. Typical parallel programming models combine sequential languages with a message passing layer, a thread library or parallelizing compilers.

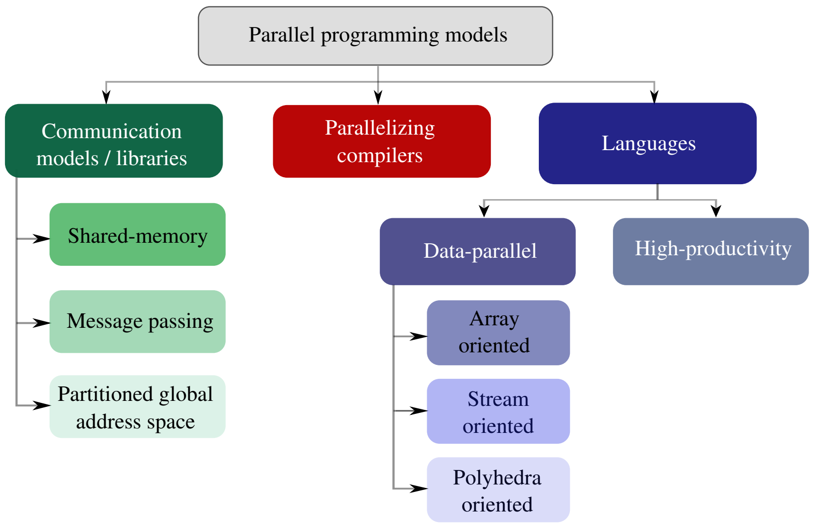


Figure 12: A glance at parallel programming models and language families.

The next paragraphs discuss the different parallel programming models and language families summarized in Figure 12.

SHARED MEMORY AND MESSAGE PASSING MODELS. Among parallel programming models, the *shared memory* and *message passing* models appear as the mainstreams. The *partitioned global address space* model inherits from both models. We briefly present each of these models.

The shared-memory programming model defines a framework where parallel computation threads communicate via shared variables. The access cost of these variables is uniform[2], meaning that no notion of memory proximity is explicitly distinguished. Typical parallel programming languages that rely on this model are OpenMP [195], Posix threads [190] and Cilk [212]. OpenMP, the most representative of them, defines an application programming interface (API) composed of a set of compiler directives, library rou-

1 An *execution model* [152] differs from a programming model by defining the interaction between physical and abstract objects that actually achieve the computations. Concretely, an execution model provides the continuum between the different layers (application, runtime and operating system, and hardware) involved in a computing system. It determines how a program executes. Examples of parallel execution models are vector, SIMD and systolic machines. Further details about the definition of an execution model in high-performance computing are found in [12].

2 From the architecture point of view, the way processors get access to shared memories can be either *uniform* or not. In uniform memory access (UMA), the memory access time does not depend on a requesting process or considered memories. In *non uniform* memory access (NUMA), the access time depends on the distance between processor and memory.

tines, and environment variables. When programming in OpenMP, a user does not have to care about data layout, which is managed by a compiler.

The message passing interface (MPI) [183] specifies a set of routines to achieve communication and synchronization for parallel programming using the message-passing model. MPI aims at distributed memory machines. Its routines facilitate the data transfers between different locations. Distributed memory machines are known to be very scalable compared to shared-memory ones, which offer only a limited communication bandwidth. Further advantages of message passing over shared-memory are discussed in [155]. Existing implementations of MPI specification are OpenMPI [88] and MPICH [46]. Beyond parallel and distributed programming, the message passing paradigm is also used in other concurrent programming, such as object-oriented programming.

The partitioned global address space (PGAS) model borrows features of shared memory threads and message passing. Variables share a common address space, and are accessible to all processes. However, the address space is logically partitioned in such a way that a notion of proximity to a particular memory section is taken into account for processes. The advantage of this vision is to provide the necessary locality information for efficient and scalable mappings of data structures onto both shared and distributed memory hardware. Examples of languages adopting the PGAS model are the language extensions Unified Parallel C [49], Co-Array Fortran [193], and Titanium [243] for Java.

DATA PARALLEL AND HIGH PRODUCTIVITY LANGUAGES. There is a wide variety of parallel programming languages, generally defined for a specific purpose. The next paragraphs overview two families of languages, which have common motivations with our RSM formalism.

Data parallel languages differ from traditional sequential languages in their use of operations and assignments over aggregate data structures, typically arrays. They are quite popular on SIMD architectures[3] such as Connection Machine (CM-2) [233] and the Massively Parallel Machine (Maspar), in which the hardware supports fine-grained parallelism.

Array-oriented languages. APL (A Programming Language) [140] is a pioneer array-oriented language for the design of digital computing systems at a high abstraction level. It proposes a concise notation to describe operations on arrays, without explicit loop definitions. High-performance Fortran (HPF) [136] also uses a compact notation for parallel loop constructs and regular data distributions. NESL [38] provides constructs for expressing nested data-parallelism concisely and a performance evaluation model. It is well-suited for irregular algorithms manipulating graphs and sparse matrices.The Single Assignment C (SAC) language [218] is a functional language manipulating arrays to describe intensive signal processing applications. It is similar to the Array-OL domain-specific language [72, 121], which is dedicated to regular multidimensional signal processing applications.

Stream-oriented formalisms. The synchronous dataflow (SDF) model [166] consists of a set of nodes communicating via FIFO queues. The rates of exchanged monodimensional data tokens are specified statically. SDFs

3 Notice that these architectures are no longer adopted nowadays. Instead, SIMD extensions of instruction sets or GPUs are found.

are well-suited for static analysis and scheduling. Kahn process networks (KPNs) [149] are another popular mathematical design model. They consist of processes representing (sequential) programs that communicate via *unbounded* FIFO buffers with blocking read. They define deterministic specifications and offer a high flexibility for compositional design. Further relevant design models include marked or event graphs [61], which are a sub-class of Petri nets [202]. Data parallel applications are also defined with the single assignment Sisal functional language [107] and the Brook [222] extension of C language, which provide an implicit expression of parallelism in terms of streams and iterations. The StreamIt [229] programming model, dedicated to streaming applications, provides a compiler for an efficient execution.

Polyhedra-oriented formalisms. Beyond the above specification models, we also mention *Alpha* [182, 240], a functional language devoted to the expression of regular data-intensive algorithms. Alpha relies on systems of affine recurrence equations. The manipulated data types consist of convex polyhedral index domains. This allows for the description of algorithms operating on multidimensional data. Alpha is very close to the Array-OL language [115]. It is associated with a few program transformations for optimal implementations. Polyhedral process networks (PPNs) [20] are variants of KPN models, where communicating FIFO buffers hold a bounded size, with blocking reads and writes. The executions and inputs/outputs of PPNs are described as polytopes resulting from a polyhedral analysis of static affine nested loop programs. PPNs have been used to generate task and pipeline parallel programs for embedded architectures.

High-productivity languages aim to facilitate the programming of high-performance systems by providing high level language supports for abstraction and modularity, using for instance object-oriented programming, together with new ideas for describing massive parallelism. They share common features with PGAS: global name space and explicit representation of locality. However, they also allow dynamic creation of parallelism, which is necessary for achieving operations on irregular data structures. Examples of languages are Chapel [52] developed by Cray, Fortress [9] designed by Sun and X10 [55] a Java extension defined by IBM. More recently, the CUDA programming model [191] has been introduced by Nvidia as a solution to facilitate programmers' task for implementing scalable parallel applications on massively multithreaded GPUs. It consists of an extension of sequential programming languages such as C with abstractions allowing for the expression of parallelism.

AUTOMATIC PARALLELIZATION. The "90–10 rule" is a well known empirical fact in software code optimization community that suggests 90% of execution time of a program comes from 10% of the code. Most often, this 10% code fragment is composed of loop nests for which the inherent parallelism must be exploited in an adequate way for an efficient execution. Automatic parallelization [21] has been largely studied in order to permit the efficient parallel execution of code fragments written, typically in a sequential language such as C, without modifying programs. There have been interesting results for programs manipulating arrays and regular nested loop [43]. Several existing compilers implement these techniques, e.g., SUIF compiler [13], Intel C++ compiler or LLVM [157]. While automatic parallelization has been successfully applied to shared memory machines, for distributed memory machines the problem is more complex. As observed by [152], the "*ability to automatically extract from serial programs more operations out of normal programs to perform in parallel in the hardware has plateaued*". Among difficulties in automatic parallelization are pointer handling and dynamic control management. Beyond automatic parallelization, there are further techniques such as allocation, scheduling or memory management of loop nests, which contribute to efficient execution of programs.

3.2.2 *The repetitive structure modeling (RSM)*

The domain-specific language Array-OL [72] [115] has been initially proposed within an industrial context by Thomson Marconi Sonar (now Thales) for specifying regular multidimensional signal processing applications. The complexity of such applications does not come from the managed elementary functions, but from the way the functions have access to data in multidimensional arrays. These elementary functions usually consist of sums, dot products or Fourier transforms, which are often available in optimized library implementations. The complex data access patterns in concerned applications make difficult their scheduling on parallel and distributed execution platforms. The real-time constraints on these data-intensive applications call for a suitable exploitation of their inherent potential parallelism on parallel hardware.

Starting from Array-OL, the DaRT group at LIFL and Inria (Lille) proposed an extension for a more general modeling paradigm, referred to as *repetitive structure modeling* (RSM) [106], dedicated to the co-modeling of massively parallel systems, from different viewpoints: application functionality, hardware architecture and functionality/architecture mapping.

We have proposed a tentative operational semantics for RSM in [105] (not presented here).

BASICS OF RSM. Among the basic characteristics of RSM are the following: *true data dependency expressions, determinism, absence of dependency cycles, hierarchy, single assignment* and *undistinguished temporal and non temporal dimensions in toroidal multidimensional arrays*. An application is specified as an oriented task graph in which three kinds of tasks are distinguished: *elementary*, *repetitive* and *composed* tasks. These tasks have input and output *ports*.

An **elementary task** is an atomic (or sequential) function. A **repetitive task** expresses data-parallelism by specifying how a given task is repeated on different data subsets, referred to as *tiles*. It is illustrated in Figure 13. Its associated *repetition space* $\mathbf{r}$, a vector in which coordinates are iteration bounds, gives the total number of repetitions. Input and output arrays are conveyed by white square ports. Each tile, conveyed by a dark square port, is processed by a repeated task instance corresponding to some task. All ports are associated with a *shape* information representing the static size/dimension of their conveyed arrays and tiles. In Figure 13, the shape of the unique input array port is a $(8,7)$-matrix. The absolute coordinates of every tile within an array are computed via information provided in a specific connector, called *tiler*, which connects an input/output array port to a corresponding tile port.

A tiler specifies the following information: the coordinates $\vec{O}$, referred to as *origin vector* of the data array; a *paving matrix* P used to compute the absolute coordinates of tile origins in an array; and a *fitting matrix* F used to compute data coordinates within a tile.

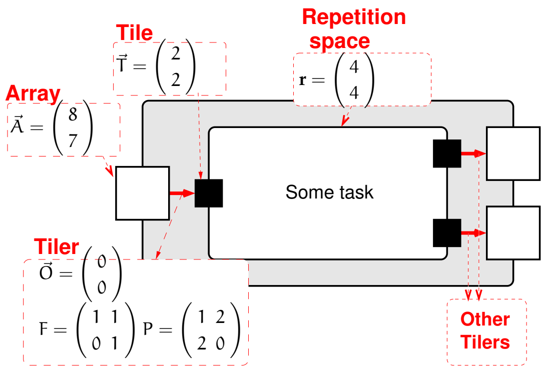

Tile
$\vec{T} = \begin{pmatrix} 2 \\ 2 \end{pmatrix}$
Repetition space
$\mathbf{r} = \begin{pmatrix} 4 \\ 4 \end{pmatrix}$
Array
$\vec{A} = \begin{pmatrix} 8 \\ 7 \end{pmatrix}$
Some task
Tiler
$\vec{O} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$
$F = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$ $P = \begin{pmatrix} 1 & 2 \\ 2 & 0 \end{pmatrix}$
Other Tilers

Figure 13: Repetitive task specification.

The repetition space $\mathbf{r}$ of a repetitive task is parsed by a vector index $\mathbf{q}$. For instance, in Figure 13, such an index is bidimensional $\mathbf{q} = \begin{pmatrix} i \\ j \end{pmatrix}$. The paving matrix P is used to compute the coordinates of *the origin point* for each tile processed by a repeated task instance, identified by index $\mathbf{q}$. Such an origin is a point in the array $\vec{A}$, defined by:

$$\vec{T_{\mathbf{q}}} = (\vec{O} + P\mathbf{q}) \bmod \vec{A} \qquad (1)$$

where the index $\mathbf{q}$ is defined on the $\mathbf{r}$ space, i.e., $\mathbf{0} \leqslant \mathbf{q} < \mathbf{r}$.

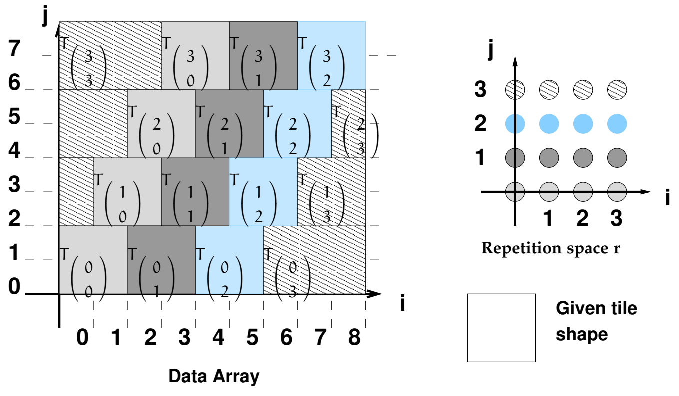


Figure 14: Array paving according to a repetition space.

Figure 14 illustrates the paving of the input data array for the previous repetitive task given in Figure 13. It shows the layout of tiles and the covered repetition space $\mathbf{r}$. For each repetition in $\mathbf{r}$, a tile is computed in the array $\vec{A}$ and the correspondence between tiles and repetition indexes is shown via colors. For example, during the repetition identified by $\mathbf{q} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$ in the $\mathbf{r}$ space (right part of Figure 14), the origin point of the corresponding tile $\vec{T_{\mathbf{q}}}$ in the array (left part of Figure 14) is computed as:

$$(\vec{O} + P\vec{q}) \bmod \vec{A} = P\vec{q} = \begin{pmatrix} 1 & 2 \\ 2 & 0 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} \begin{pmatrix} 2 \\ 0 \end{pmatrix}.$$

After the identification of tile origin points, one has to determine how to fill each tile with the elementary data contained in an array. For this purpose, an intra-tile repetition space $\vec{T}$, defined as the shape of a tile, is considered. It is parsed by an index $\mathbf{p}$. The positions of elementary data into a tile are computed by using the fitting matrix F, as follows:

$$\vec{d_{\mathbf{p}}} = (\vec{O_{\mathbf{q}}} + F\mathbf{p}) \bmod \vec{A} \qquad (2)$$

where $\mathbf{o} \leqslant \mathbf{p} < \vec{T}$ and $\vec{O_\mathbf{q}}$ is the origin of a tile.

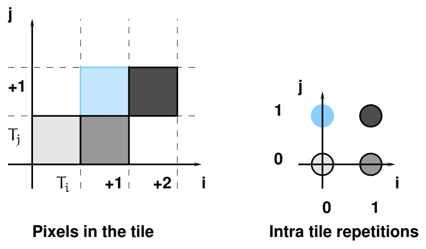


Figure 15: Data layout inside a tile given by the fitting matrix F.

Figure 15 shows a correspondence between the intra-tile repetitions and their associated elementary data. The correspondence is highlighted by colors.

Within the same repetition space, task instances may depend on each other. This is typically useful when computing the integral sum of array elements. Such a repetitive task belongs to the extension of Array-OL defined by the DaRT group (called "Array-OL with delays") and is referred to as *repetitive task with inter-repetition dependency* [121]. Figure 16 illustrates such a task connecting the output tile port of each repeated task instance to the input port of its dependent repeated task instance, according to a dependency vector $\vec{d}$. This vector indicates within the repetition space the uniform dependencies between repetition instances. A specific connector, denoted by $\mathtt{init\_val}$, specifies initialization tile values for repetition instances with dependencies that are out of the repetition space. These instances are typically the very first ones according to the order introduced by an inter-repetition dependency.

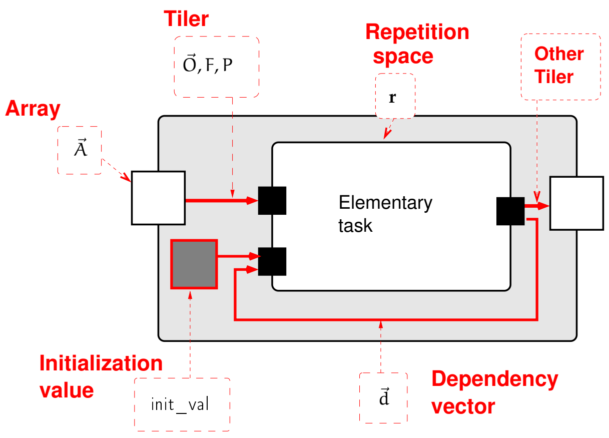


Figure 16: Repetitive task with inter-repetition dependency.

A **composed task** is defined by a directed acyclic graph (DAG) composition of elementary and repetitive tasks, allowing for task parallelism representation. Figure 17 shows an example of RSM specification, composed of four interconnected repetitive tasks.

LOOP TRANSFORMATIONS IN RSM. Many loop transformations are usable to modify RSM descriptions [114]. Below are presented some of them.

Fusion. Let R_1 and R_2 denote two repetitive tasks exchanging an array A produced by R_1 and consumed by R_2. The computations of task R_2

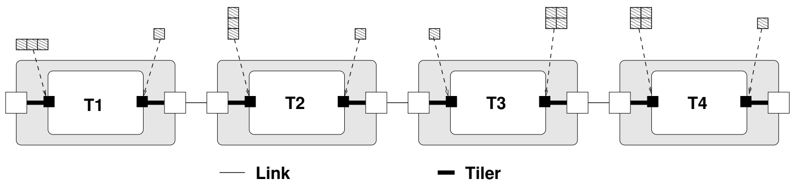


Figure 17: An Array-OL specification composed of four tasks.

may be started while only sub-parts of A are produced by R_1. Let us call macro-tile such an array sub-part. An execution at the macro-tile level enables a pipelined execution of R_1 and R_2. It also allows to minimize the size of intermediate memory required for data storage between the tasks. The fusion transformation of tasks R_1 and R_2 creates a new repetition space on top of R_1 and R_2 such that: i) the exchanged array A is replaced by a macro-tile and, ii) the repetition spaces of R_1 and R_2 are reduced so as to produce and consume these macro-tiles instead of an array with the same shape as A. So, the fusion changes the granularity of exchanged data and task repetition spaces.

Paving change. Let us consider a hierarchy of repetitive tasks, equivalent to a loop nest. There are repetition spaces at different levels in this hierarchy. The paving change transformation enables to redistribute the repetitions between the different levels by moving part of the higher level repetitions to lower level ones. For instance, for a given initial specification consisting of a hierarchical repetitive task R_1 where the inner repeated task is itself a repetitive task R_2. Let their respective repetition spaces be $[t, s]$ and $[c]$. A change paving transforms the specification so that the higher level repetition space becomes $[t]$ and the lower level one is $[s, c]$. If dimensions t, s and c denote respectively time, space and computation, it means from the initial specification, one is making explicit the temporal execution flow of a functionality computed in R_2.

Collapse (unrolling). Given a hierarchy of repetitive tasks as discussed in the previous transformation, the collapse transformation is equivalent to paving change operation applied to all higher hierarchy levels. The corresponding repetition spaces therefore become empty and useless, except the repetition space at the lowest internal level. As a result, all hierarchy levels associated with empty repetition spaces are deleted from the specification. For instance, for a hierarchical repetitive task R_1 where the inner repeated task is itself a repetitive task R_2, if their respective repetition spaces are $[t, s]$ and $[c]$, an application of the collapse transformation will yield a (non hierarchical) repetitive task R_2 associated with a repetition space equals to $[t, c, s]$.

Tiling. Given a repetitive task, the tiling transformation performs the inverse operation of the collapse. It creates a new repetition level on top of a given repetitive task. In other words, this adds a new hierarchical level in the specification. During this transformation, the repetition space of the initial repetitive task is divided into sub-spaces. In the resulting task specification, the repetition space associated with the added level (i.e., higher hierarchy) expresses repetitions between the sub-spaces, while the repetition space corresponding to the initial level (i.e., lower hierarchy) expresses repetitions within a sub-space.

Figure 18 illustrates an Array-OL specification derived from the specification in Figure 17 by applying the following transformations: *i)* fusion of tasks T1 and T2, where the two elementary tasks T1 and T2 are merged in a common repetition space; *ii)* tiling of task T3, where a new repetition hierarchy level is created and the repetitions are distributed between the hierarchy levels; and *iii)* paving change of task T4, where the size of the consumed and produced tiles are increased.

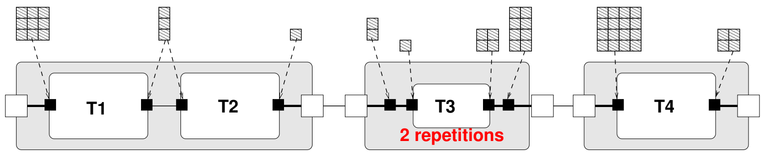


Figure 18: Specification of Figure 17 after 1) fusion of tasks T1 and T2; 2) tiling of task T3 and 3) paving change of task T4.

Beyond all above notions, there are other extended concepts of RSM, proposed by the DaRT group, for the description of regular hardware architecture topologies, and regular mappings of software applications on hardware platforms in codesign. These concepts have been integrated in the Marte standard profile [194] of the Object Management Group (OMG). This profile is dedicated to the *Modeling and Analysis of Real-Time and Embedded systems*.

In addition to our Gaspard2 design environment [106] [71], there are other environments which also adopt Array-OL concepts like the SpearDE codesign framework of Thales Research & Technology and the Ptolemy II of Berkeley via its very recent *Pthales* domain [22].

3.3 FROM STATIC TO DYNAMIC DESIGN MODEL IN RSM

The works presented in this section have been started from September 2005, when I started my Post-doc in the DaRT group. Part of them also covers the PhD thesis of Huafeng Yu (October 2005 – December 2008).

Huafeng Yu is now Research Engineer at INRIA Rennes (France).

3.3.1 *Integrated control-oriented design with FSMs and RSM*

SOME RELATED WORKS. The Mode Automata formalism [179] integrates FSMs and the Lustre language to enable a direct specification of complex systems based on the notion of running mode. This provides a designer with dataflow and imperative flavors at the same time. More recent variants and extensions of Mode Automata have been defined in the Lucid Synchrone [60] and Signal languages [225]. In Signal, a previous work [215] studied the combination of a preemption mechanism with the associated multi-clock dataflow model by considering clock activation periods.

Similar extensions are introduced in SDFs to express dynamic changes and reconfiguration in streaming applications [227] [189]. These solutions integrate new features to SDFs in order to describe system behavior modes or scenarios. In the major part of solutions, FSMs are used to define the control part as reported in [82]. For instance, the ∗charts family of models of computation proposed earlier in Ptolemy [112] makes it possible to combine hierarchical FSMs and dataflow graphs. The states of an FSM can be refined as dataflow graphs while actors of a dataflow graph can be refined by FSMs. The applicability of this design principle of ∗charts is limited to dataflow graphs for which execution can be divided into finite iterations, i.e.

a minimal number of actor firings setting a dataflow graph to its initial state. Extended codesign finite state machines (ECFSMs) [217] are finite state machines that manage the communication behavior of an actor in a network where interconnects are FIFO channels. This model has been extended with the notion of actor state and hierarchy in SysteMoC [133]. The California actor language (CAL) [79] defines dataflow graphs where actors can have state information provided by FSMs dedicated to the scheduling of actions to be executed.

There are some methodologies devoted to integrated FSM and dataflow design such as Ptolemy II [80], OpenDF Design Flow [36], SystemCoDesigner [151] and Windowed Data Flow (WDF) [150]. The most popular is undoubtedly the former, Ptolemy, which integrates several models of computation (e.g., finite state machines, synchronous dataflow, concurrent sequential processes, process networks...) in order to provide an environment for heterogeneous design. WDF aims to multidimensional applications.

The previous integrated design approaches are closely related to multi-formalism programming models, which are used for hybrid discrete/continuous system design. We have already mentioned the Ptolemy framework, which allows for that. Matlab [138] enables to describe modes in event-driven and continuous systems by using Stateflow specifications. Hybrid Sequence Charts (HySCs) [120] are a specialized subset of Message Sequence Charts (MSCs) providing a visual description of discrete and continuous behaviors. Finally, a language [30] has been defined recently for hybrid system modeling from a dataflow synchronous language. It integrates hierarchical automata combined with dataflow and differential equations.

MODE TASKS AND TRANSITION FUNCTIONS. A reactive control modeling is associated with RSM in the form of an extension relying on *finite state machines* (FSMs) [156] [244]. More precisely, the connection between the data-intensive and control parts of an application is defined by distinguishing data computation modes and transition functions that produce control values to select some modes. One important requirement here is to preserve the regularity of computations expressed in RSM while they are made controllable.

Key publications for more details: [209] [245].

In order to enable the description of data-intensive applications including control-oriented behaviors, the RSM model has been extended [156] with a reactive control notion in the form of Mode Automata [179]. With Éric Rutten and Huafeng Yu, we have enhanced this initial proposition by generalizing the mode automata notions in this context by making possible descriptions featuring hierarchical and parallel mode automata [244]. Note that this enhancement is prior to the definition of the new *Pthales* domain [22] in Ptolemy. This domain certainly opens interesting opportunities in terms of heterogeneous designs, and particularly regarding the interaction between control-oriented models of computations and data-intensive computations. The next paragraphs give an overview of the main concepts in our proposition.

A **Mode task** expresses a choice among several alternative computations denoted by tasks T_j, also called *modes*. All the modes T_k of M have the same interface. Figure 19 illustrates a mode task in an informal notation inspired by windows with multiple tabs [156]. The task is composed of four modes $T_0 \dots T_3$, each identified by a mode value: $m_{i, i\in 0..3}$. It has a specific input port m, called *mode selector*. When m holds the value m_k, the computation performed by the mode task is that of T_k. As in Mode Automata, the modes run in mutual exclusion, meaning that whenever a mode task executes, only the task T_k associated with the selected mode m_k is computed.

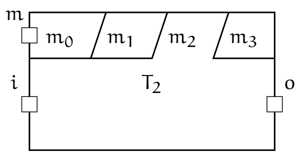


Figure 19: Example of mode task.

Transition functions are used to define mode values serving to select computations. Given some inputs c used in transition conditions and a current state value s, such a function is an elementary task in RSM that computes the next state value s' after transitions. To define an automaton, a transition function task is embedded in a repetition as depicted by Figure 16. An inter-repetition dependency vector $\vec{d} = (-1)$ is considered over a totally ordered dimension of the repetition space **r**, e.g., a monodimensional temporal dimension in **r**. This vector connects the ports associated with state values s and s'. This automaton encoding is very similar to the usual encoding of automata in sequential circuits. Advanced models such as hierarchical or parallel automata can be obtained in a similar way [101].

Hierarchical constructs can be defined via a simple extended data dependency model that mixes data-intensive computations with control. This model has a similar semantics as a mode automaton [179]. The statements representing the data-intensive part are executed according to the states computed by a transition function. Figure 20 shows a macro construct consisting of a repetitive task R with an inter-repetition dependency. The repeated task is a hierarchical task H in which, a mode task selects a data-intensive algorithm to compute the resolution of images, according to a power level.

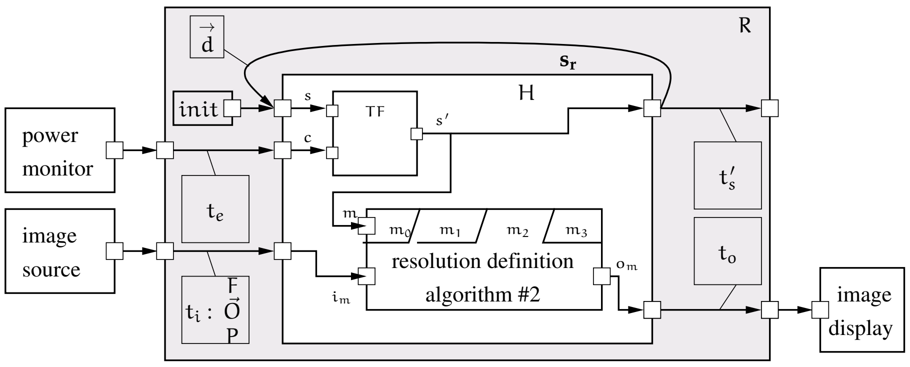


Figure 20: Example of mode automaton.

APPLICATION TO MPSOC CO-MODELING. The above control-oriented extension of RSM has been experimented at various levels in our co-modeling framework Gaspard2. In particular, we showed that the extension is generic enough to be used for modeling of software application, hardware architecture, association of both and deployment (i.e., instantiation) with Intellectual Properties (IPs).

While I personally have been mainly involved in the software application level, I got the chance to collaborate with colleagues in the DaRT group on the usage of the extended model in other levels. More precisely, we addressed together the generation of hardware accelerators for dynamic reconfiguration on FPGAs in Gaspard2 [209, 70, 210].

Currently, within the French ANR project, named Famous, in which I am involved, our gained insights are being considered for an extension of the Marte standard profile for reconfigurable systems, referred to as "Reco-Marte" profile.

3.3.2 *Interaction between data dependencies and logical time*

Now, I address a refinement of the untimed computation and communication actions expressed by data dependencies in RSM into synchronous reactive models in which the temporal dimension of computations is made explicit. For this purpose, I studied a structural translation of RSM specifications into synchronous models following their syntactical constructs. This translation is greatly facilitated by the similarity between RSM and synchronous dataflow languages since both have a recursive block-diagram structure. It is summarized in this section after an overview of some related studies.

SOME RELATED WORKS. A reference work combining data-intensive computation and abstract clocks of synchronous languages is [219], in which authors consider the functional data parallel language Alpha [182, 240, 164] and Signal. In their approach, intensive numerical computations are expressed in Alpha while the control (the clock constraints resulting from Alpha descriptions after transformations) is conveyed to Signal. The regularity of Alpha enables to identify affine relations between the specified clocks. The Signal compiler therefore addresses synchronizability criteria based on such clock relations. Similar concepts can be found in [59] where a synchronous model is defined in order to address the correct development of high performance stream-processing applications. It particularly relies on a domain specific knowledge consisting of periodic evolution of streams. This model allows to automatically synthesize communications between processes with periodic clocks that are not strictly synchronous.

Although synchronous dataflow languages are not specifically intended for data-intensive computations, they include some interesting features such as arrays and iterators. Arrays have played an important role in synchronous dataflow programming for the description of algorithms and architectures. We mention the pioneer work of Le Guernic, Benveniste, Gautier and Bournai on the definition of *regular arrays of processes* in the Signal language for signal processing applications [160] [161]. This enabled the description of regular data-parallel algorithms.

Another pioneer work in the Lustre language concerned the introduction of arrays in the language [129] by Halbwachs and Pilaud. This work concentrated on the simulation of systolic algorithms. The authors showed that arrays are necessary in order to write systolic algorithms in Lustre. These array structures have been implemented on FPGA by Rocheteau [214]. More recently, Morel proposed an efficient compilation of arrays in Lustre programs [185]. Another work on Lustre arrays [128] involves the array content analysis through abstract interpretation.

Other relevant languages such as StreamIt [229] and Otto E Mezzo [184] can be mentioned. The former has been already introduced in Section 3.2.1. The second allows to describe behaviors of dynamical systems. It uses clock information in the code generation, e.g., in C or towards a SIMD abstract machine. It is inspired by the multidimensional extension [204] of the Lucid dataflow language of Wadge and Ashcroft [238].

SPACE-TIME MAPPING FOR LOGICAL INSTANT IDENTIFICATION IN RSM. As a basic principle, we decide to map the infinite dimension of repetition spaces in RSM onto temporal dimension. The loop transformations introduced earlier enable a re-factoring of RSM models into a form that can be straightforwardly interpreted over a temporal dimension. Such a form consists of a hierarchical task in which infinite arrays are manipulated only at the very top-level. In this task, the top-level is composed of a single task playing a similar role as a "`main`" function in a C program.

According to a chosen instant granularity, the main task at the very top-level instantaneously computes identical sub-parts of input infinite arrays, as in a flow of arrays. For instance, in a video processing application, one may need to consider that input video data are read image by image, or by sets of images. Then, the granularity of an instantaneous reaction is the processing of either an image or a set of images as illustrated in Figure 21.

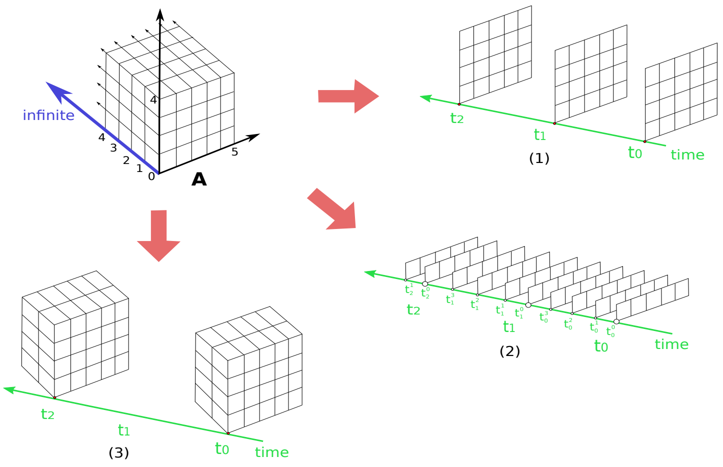
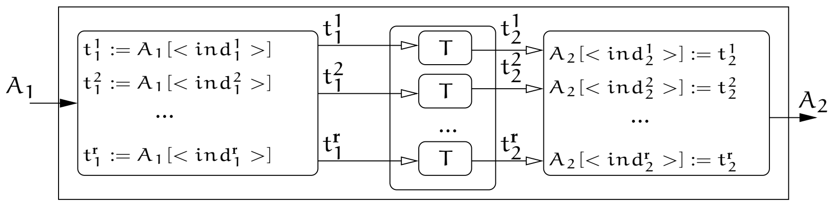


Figure 21: Space-time mapping of a $[5, 4, \infty]$-array w.r.t. different granularities.

In [11], the authors adopted the same approach to define a projection of data-parallel applications specified in Array-OL onto KPNs. They considered a pipelined execution on data streams resulting from a refinement of manipulated infinite arrays. The fusion loop transformation has been applied to set application models in the right form.

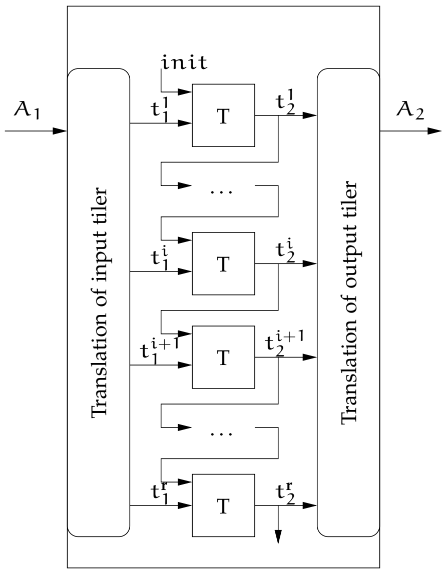
Understanding the space-time mapping issues in a context combining both RSM and synchronous reactive modeling paradigms was part of a two-year collaboration between the Inria Aoste, DaRT and Espresso groups. This collaboration, named Triade[4], focused on SoC design. Its aim was to explore a seamless flow of increasingly time-defined and time-accurate models, so as to progressively derive implementations through provably correct steps from high-level (loosely-timed) models. In a submitted joint-paper with members of Aoste, we present how from loop transformations applied to RSM models according to given environment and execution platform con-

4 http://www.irisa.fr/espresso/Triade



$t_1^1 := A_1[< ind_1^1 >]$
$t_1^2 := A_1[< ind_1^2 >]$
$t_1^r := A_1[< ind_1^r >]$
A_1
T
$A_2[< ind_2^1 >] := t_2^1$
$A_2[< ind_2^2 >] := t_2^2$
$A_2[< ind_2^r >] := t_2^r$
A_2

(a) Task *without* inter-repetition dependency.



(b) Task *with* inter-repetition dependency.

Figure 22: Parallel synchronous models of repetitive tasks

straints, temporal and scheduling properties are captured via specifications defined in the polychronous CCSL language. The PhD thesis of Coadou [57] shares a few similar motivations with this work. It considers k-periodically routed graphs and polyhedral models to deal with loop schedulings for data-intensive processing.

FROM RSM MODELS TO SYNCHRONOUS DATAFLOW PROGRAMS. Provided the previous time-space mapping is applied to a given application, we can translate RSM in synchronous dataflow languages. Basically, each RSM task is represented by a Signal process (or Lustre node equivalently), with the same interface. Ports are translated as signals. An elementary task is represented by a function. Composed tasks are translated in an inductive way by using the composition operation of synchronous languages on the translation of its component tasks. A repetitive task R is translated similarly by composing the translations of its tilers and repeated task instances T, as follows (see Figures 22a and 22b):

Key publication for more details: [102].

- for an input tiler $(F, \vec{O}, P)$, the corresponding Signal process takes as input an array with shape $\vec{A_i}$ and produces tiles t_i^q via a set of equations enumerating the extraction of the tiles. The index ind^q corresponding to the $\mathbf{q}^{th}(\mathbf{o} \leqslant \mathbf{q} < \mathbf{r})$ tile having the shape $\vec{T}$ is obtained as follows:

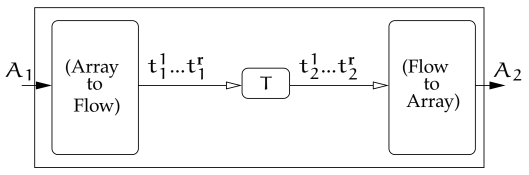
$$< ind^q >= \{\vec{O} + \mathbf{q}P + \mathbf{p}F \bmod \vec{A}, \text{ where } \mathbf{o} \leqslant \mathbf{p} < \vec{T}\} \quad (3)$$

For output tilers, the corresponding Signal process takes as inputs some tiles and produces an array in which the tiles have been stored at suitable indexes according to above formula 3.

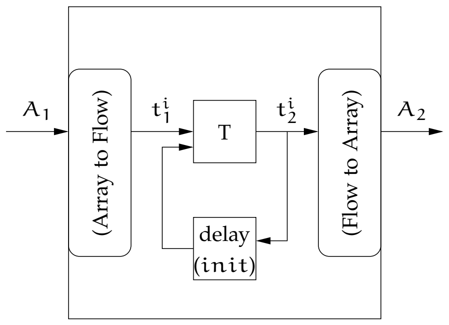
- the set of repetitive task instances executed in parallel is encoded by a Signal process consisting of the parallel composition of $|\mathbf{r}|$ identical translations of the repeated task T in R. When R contains a dependency between repetitions, the associated synchronous model includes additional dependencies between encoded repeated tasks. The initialization values are inputs of the first instances w.r.t. dependency order.

In addition to the above translation, we have also defined an alternative encoding that serializes the execution of a repetitive task R. The resulting synchronous model is more compact since it does not enumerate all instances of R. This contributes to mitigate scalability issues in the resulting synchronous models. Roughly speaking, this new translation is as follows (see Figures 23a and 23b):

- input and output tilers are respectively encoded by special Signal processes, referred to as *Array to Flow* and *Flow to Array*. The former process applies an oversampling to input arrays to extract a flow of tiles. The latter process conversely applies a down-sampling on a flow of tiles to construct an array.
- the inner task T in R is represented by a Signal process encoding one instance of R processing flows of tiles. For repetitive tasks with dependency between repetitions, the associated synchronous model includes a delay operation on flows of tiles.



(a) Task *without* inter-repetition dependency.



(b) Task *with* inter-repetition dependency.

Figure 23: Serialized synchronous models of repetitive tasks

For more compact synchronous models in our translation, another solution would be to exploit extended features of synchronous languages. The arrays and their associated iterators added in Lustre [185] and the *array of processes* construct [33] of Signal can be considered for this purpose.

Indeed, the synchronous model encoding presented here is meant to be intuitive and simple, and may certainly be optimized. However, the considered optimizations can be quite different according to the goals, e.g., efficient code generation or verification. Our translation separates assignments to array elements in different equations. This enables to apply causality analysis at array element level.

The translation of the RSM extension with FSMs is quite similar to the previous encoding rules. A mode task is simply translated as a "case" statement. When such a construct is not provided by a language, it is encoded as a composition of conditioned equations, each corresponding to a possible mode. Whenever a condition is evaluated to *true*, the equations associated with the evaluated mode are chosen for computation. A transition function is encoded in a very similar way, or by considering built-in automata constructs when available in languages.

LOOP TRANSFORMATION-BASED ABSTRACTION FOR SCALABILITY. In a collaboration referred to as ID-TLM , initiated in December 2008 (for three years) between ST Microelectronics and Inria, we began a study on the modular design of massively parallel applications based on component-based abstraction. This was also the subject of the short Post-doc fellowship of Mohamed Fellahi. Here, I highlight our most relevant results [117] regarding this project, obtained together with Pierre Boulet and colleagues from the Aoste group of Inria and I3S (Sophia Antipolis).

Accepted publication on this topic: [117].

To deal with the scalability issues in our previous translation, we investigated an abstraction of the potential parallelism expressed in RSM. The abstraction of a task component $\mathcal{C}$ is built using a bottom-up process starting from elementary components used as building blocks of $\mathcal{C}$ up to the top level composition of $\mathcal{C}$. At each level, the process uses re-factoring transformations from the set of loop transformations described in Section 3.2.2.

At the lowest hierarchical level, the abstraction of an elementary component E is a single degenerate data-parallel repetition of the component itself. Since, there is no parallelism, the execution of component is atomic. For a repetitive component R, the main question is how to build the abstraction for a repetition of an abstraction of an internal repeated component. As this internal repeated component is abstracted itself by a data-parallel repetition, the problem is viewed as how to abstract two nested data-parallel repetitions into a data-parallel repetition. This problem is exactly solved without any loss of parallelism by applying the collapse transformation so as to obtain a component with a single repetition level on top of a graph of components, referred to as *flatten* transformation.

To build the abstraction of a DAG component C of abstracted components, we create a new hierarchy level where the top-level component is a data-parallel repetition of a DAG of the original abstracted components. The top-level repetition represents a factorization of the parallelism expressed by the repetitions of the abstracted components as in loop fusion. Actually, this transformation is a succession of fusion and above *flatten* transformations on any two components at a time in the DAG C. The final abstraction of C is obtained by keeping *only the top-level repetition* as a way to express part of the potential parallelism of C. This process may loose some potential parallelism but keeps the parallelism that can be expressed as nested loops with uniform dependencies.

The above abstraction of internal parallelism in an RSM task component is an alternative model specification that can reduce the complexity of initial

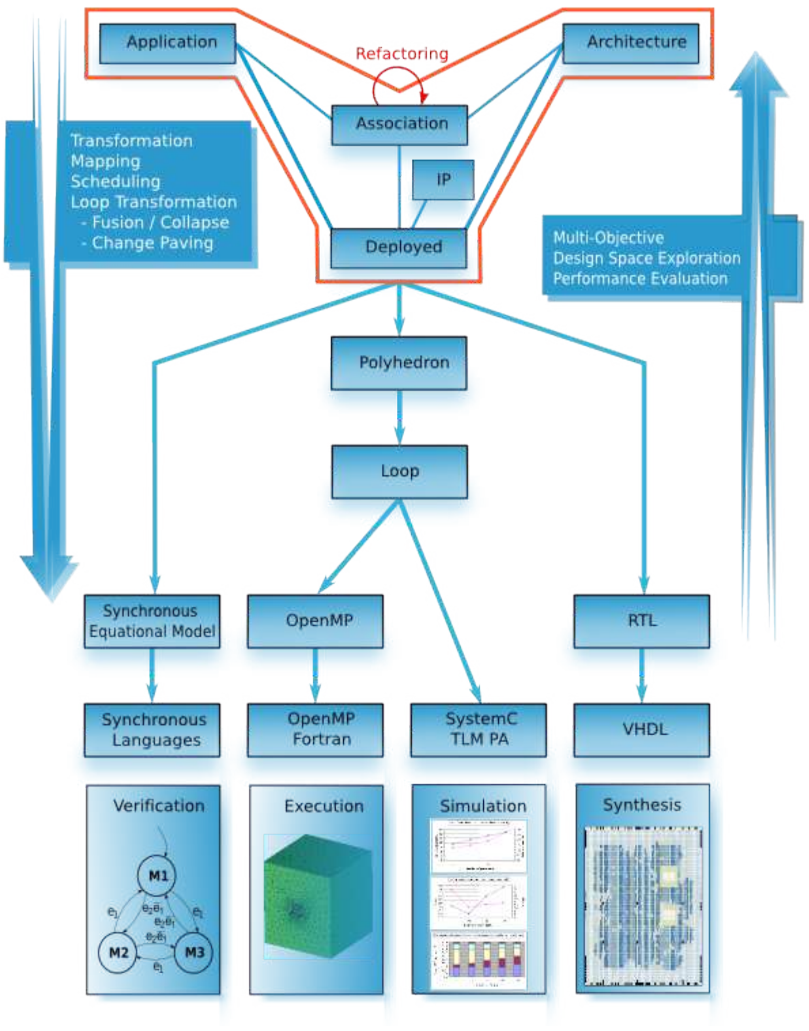


Figure 24: Sketch of the Gaspard2 design approach.

specifications. It also offers a rich interface, as introduced by Alfaro and Henzinger in [8], exposing the potential parallelism of a component while hiding its implementation. This favors component reusability in a design context where the effective use of the potential parallelism of a component is a decision that could be taken after the design of the component library.

3.3.3 *Model-driven engineering in Gaspard2 environment*

Key publications for more details: [244] [103].

Key publication for a more detailed overview: [106].

In the context of the PhD thesis of Huafeng Yu [244], a prototype transformation chain, based on model-driven engineering, has been developed in Gaspard2 (first version) for an automatic translation of RSM models into synchronous dataflow programs. It is illustrated on the left-hand side of the Gaspard2 design approach shown in Figure 24. In this approach, the comodeling of data-intensive MPSoCs starts with Marte specifications of software application, hardware architecture, association of both parts and deployment of generic components with IPs. Then, different automatic model transformations are implemented towards various technical domains, for: verification of application properties via synchronous languages, high-performance execution via OpenMP Fortran (and more recently via OpenCL), system simulation via SystemC, and hardware synthesis via VHDL.

The implementation of the transformation towards synchronous languages relies on a generic metamodel for synchronous equational dataflow languages, which targets at the same time Lustre, Lucid Synchrone, and Signal. The tool has been developed as an Eclipse plugin. The implemented transformation rules globally represent about five thousands lines of Java code in Eclipse.

The translation chain did not include the control extension of RSM. However, a metamodel has been defined as an implementation of the proposed control-oriented extension of RSM. It relies on UML state machines and collaboration diagrams combined with already existing concepts of RSM. The transformation rules associated with the extended metamodel have been specified in [244]. A manual translation of the extended RSM have been experimented on a simple example of multimedia application [245] [104].

Beyond the above contributions to Gaspard2, I would like to briefly mention another work I was involved in, about the syntactical validation of Marte models using the Object Constraint Language (OCL) [54]. This work was done with Asma Charfi, who I advised during her master internship for eight months from April 2007. It was funded by the Franco-Tunisian *Ksours* project on design and validation for reconfigurable systems. The Tunisian partner was the Computer & Embedded Systems lab at ENIS in Sfax.

3.4 SYNCHRONOUS APPROACH FOR DEALING WITH CORRECTNESS

Some analyses applicable to RSM specifications are summarized in this section. They are typical in synchronous programming and are made possible on RSM specifications via the translation presented in the previous section. These analyses should be seen as complementary techniques to those relying on polyhedra (e.g., developed in Feautrier's works [84]), which are also applicable to RSM. They become particularly useful when data-dependencies are strongly combined with control flows.

Key publications on this topic: [102] [245] [104].

3.4.1 *Causality and array assignment analysis*

CAUSALITY ANALYSIS. The execution semantics considered for RSM in Gaspard2 framework imposes that all inputs of a task are available before outputs are computed. Let us consider a hierarchical task T, depicted by Figure 25, composed of two communicating sub-tasks T1 and T2. According to this level of hierarchy, the specification is not correct for execution since T1 and T2 are inter-dependent (i.e., causality cycle).

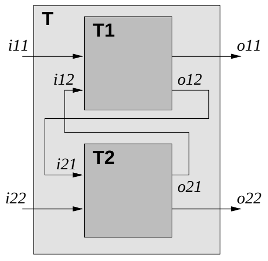
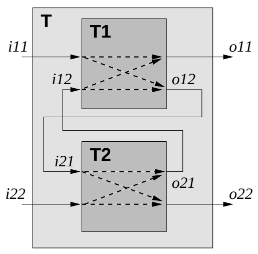
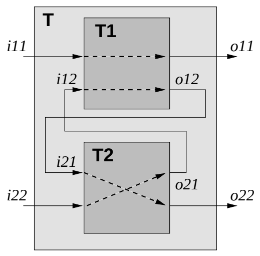


Figure 25: A simple hierarchical task model.

Our translation of RSM into synchronous dataflow languages offers the opportunity to address the causality problems inherited from RSM model with compilers associated with target languages [131] [162]. Figure 26a illustrates the task T according to the execution semantics of RSM models in Gaspard2. For instance, every output port, say o11, of task T1, depends on all input ports of the task, i.e. i11 and i12. The translation of such an RSM model leads to a synchronous program on which compilers will straightforwardly exhibit the presence of causality cycles. As a result, the version of task T given in Figure 26a should be rejected.



(a) Presence of cycle.



(b) Absence of cycle.

Figure 26: A simple causality analysis for task T.

Now, let us consider the alternative definitions of T1 and T2 shown in Figure 26b. In T1, o11 only depends on i11. Assuming the dependencies specified in Figure 26b, it is very easy to show that the translation of the second version of task T leads to a deadlock-free program. Hence, the execution semantics of RSM models in Gaspard2 clearly appears very restrictive. With a finer-grained causality analysis such as the one provided by compilers of synchronous languages, one avoids "false" data dependency cycles.

CHECKING SINGLE ASSIGNMENT. Single assignment is another key property of RSM. For a given array A, one must ensure that no element of A is overwritten after its first value assignment. This typically happens when the paving matrix and the shape of tiles lead to tiles that overlap within A. Let us consider a tiler characterized by the following values:

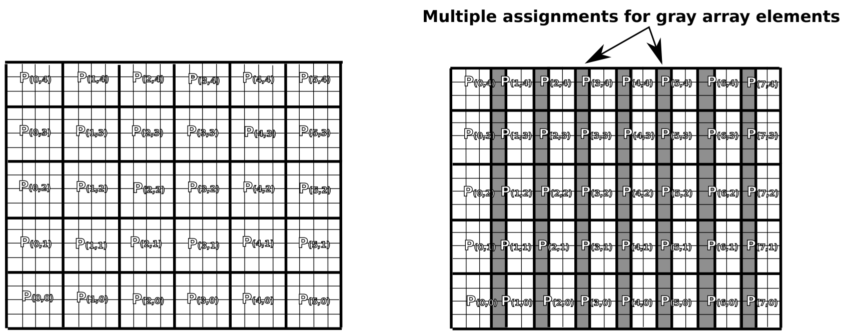
$$F = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}, \vec{O} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}, P = \begin{pmatrix} 4 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Figure 27a shows a correct paving obtained with this tiler information. The tiles have a (4,4)-shape where the origin point of each tile is set in red.

When considering new values of the tiler, given below, we observe intersection regions between contiguous tiles in Figure 27b:

$$F = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \vec{O} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, P = \begin{pmatrix} 3 & 0 \\ 0 & 4 \end{pmatrix}$$

For instance, the tile $P_{(0,0)}$ overlaps the tile $P_{(1,0)}$ at the array elements with indexes $(3,0), (3,1), (3,2)$ and $(3,3)$. The translation of RSM models with the second tiling information produces a synchronous model with multiple assignments to the same array locations. This violates the single assignment property of RSM. This is observed with compilers of synchronous languages for free.



(a) Single assignments.

(b) Multiple assignments.

Figure 27: Different array assignments in RSM: origin point in each (4,4)-pattern is represented by a red bullet.

We notice that single assignment in RSM can be checked with an algorithm that exhibits the emptiness of polyhedra intersection as shown in [41]. Such an algorithm can be implemented using linear programming. However, no implementation is currently available in the Gaspard2 environment.

3.4.2 *State-based analysis for adaptive behaviors*

Beyond the previous causality and array assignment analysis, the translation towards synchronous languages opens the way to apply further verification techniques such as model-checking of the functional properties of RSM models. This is useful when designing data-intensive applications with mode-based adaptive behaviors.

In a case study on the design of the multimedia functionality in a mobile phone, we addressed important requirements on its operating modes among which the mutual exclusion between some modes [245] [104]. Typically, black-and-white and color display modes cannot be set at the same time. This has been checked as an invariance property. Another frequent requirement is the reachability of some system modes under specific resource availability conditions. For instance, a high-quality display mode of a mobile phone may be enabled only when the battery level is high enough. To achieve the model-checking of our synchronous models, we used the Sigali tool [180].

In the same case study [104], we experimented discrete controller synthesis for the definition of a safe controller for the multimedia functionality of a mobile phone model specified with our extension of RSM. We observed that a manual encoding of such a controller, satisfying a few expected properties in a simple application, is very tedious and error-prone.

3.4.3 *Analysis in presence of environment constraints*

As discussed in the previous chapter, abstract clocks are very useful for dealing with multi-clock designs. After a space-time mapping of RSM models, they serve here to reason on applications w.r.t. environment and implementation constraints. For illustration, we consider a downscaling algorithm achieved by a component receiving a flow (t_i^k) of images from a CMOS sensor and reducing their size before sending a flow (t_o^k) of reduced images on a screen for display. This is illustrated in Figure 28. Each component is associated with an abstract clock describing its activations, i.e., its data consumption/production rates.

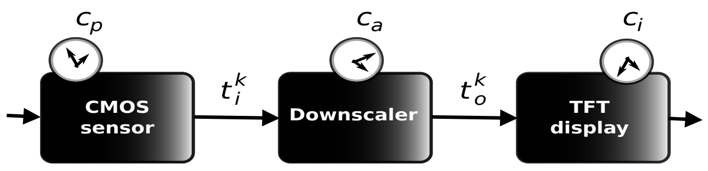


Figure 28: Image downscaling.

Let c_p, c_a and c_i respectively be the abstract clocks of the sensor, the downscaler and the display. A step in c_p, c_a and c_i corresponds to the production of respectively a single pixel by the sensor, a transformed block of pixels by the downscaler, and an image by the display. The components interaction is encoded with affine clock relations [219], as follows:

- C_1: c_a is an affine under-sampling of c_p, i.e., $c_p \overset{(1,\phi_1,d_1)}{\rightarrow} c_a$;
- C_2: c_i is an affine under-sampling of c_a, i.e., $c_a \overset{(1,\phi_2,d_2)}{\rightarrow} c_i$;

where ϕ_j and d_j respectively denote a phase and a period.

Now, we consider a design requirement of the video display functionality, consisting of a constraint on the actual image display rate, denoted by a new abstract clock c_i'. This constraint, denotes a direct affine clock relation, between c_p and c_i' as follows:

- C_3: $c_p \overset{(1,\phi_3,d_3)}{\rightarrow} c_i'$.

Guaranteeing the compatibility of C_3 with the previous set of constraints $\{C_1, C_2\}$ amounts to establish the synchronizability of clocks c_i and c_i'. Synchronizability allows to guarantee the existence of a dataflow-preserving way to make two affine clocks synchronous. In other words, a finite-size buffer protocol can be defined to synchronize such clocks. This issue cannot be addressed by only using the usual definition of clock synchronization of synchronous languages. Instead, properties of affine abstract clock systems have to be considered. From C_1, C_2 and C_3, the affine clock synchronizability property [219] implemented in the Signal compiler has to be used. This issue is solved quite easily with synchronous models while it is not possible with RSM only. Such an analysis provides feedback information for adjusting the paving iteration parameters of an RSM model of the downscaler so as to satisfy the non functional requirements imposed on the whole system.

3.5 SUMMARY AND DISCUSSION

This chapter provided an overview of my contributions on the design and analysis of reactive data-intensive applications in the Gaspard2 codesign framework. The presented results aim to bridge the gap between a repetitive structure modeling (RSM) and the synchronous reactive dataflow programming model. RSM represents applications as a hybrid of black-box stream-computing filters and regular, affine dependencies capturing their behaviors as generalized, multidimensional system of uniform recurrence equations. The synchronous model proposes well-adapted concepts to represent concurrent dataflow processes analyzable with a rich set of tools.

WHAT HAS BEEN PROPOSED? The RSM formalism does not assume any notion of time and dynamic change. Its specifications express data-dependencies between task repetitions and multidimensional access patterns, but no specific execution order is fixed. In order to deal with time and dynamic behavior changes, which are crucial notions for the description of reactive behaviors in data-intensive applications, my contributions concerned on the one hand, an extension of RSM with reactive control modeling features and on the other hand, a refinement of RSM towards synchronous dataflow languages. A major benefit of these works is *i)* to increase the expressivity of RSM for data-intensive applications with dynamic control and, *ii)* to provide a seamless flow of increasingly time-defined models in order to refine application descriptions via a translation into synchronous reactive programs. Some properties of RSM designs are therefore analyzable by using the verification tools associated with synchronous languages. These works have been conducted in the Gaspard2 environment, dedicated to the co-design of data-intensive systems-on-chip.

WHAT ARE THE MAIN LIMITATIONS AND HOW TO ADDRESS THEM? The encoding of control in RSM was achieved by considering its native features, i.e., repetitions. The motivation of such an approach is to keep on benefiting from the loop transformations already applicable to the RSM model. Another approach would be *à la* Ptolemy [22], where heterogeneous modeling enables to combine different paradigms, e.g., RSM-like with advanced control-oriented models. While it should offer more expressivity, it would have required a re-design of a large part the model transformations applicable to the RSM extension in Gaspard2.

Regarding our translation of RSM into synchronous programs, the main limitation was a scalability issue. Indeed, the size of synchronous models resulting from the transformation of RSM can be huge for large repetition spaces. This can reduce the applicability of target formal checking tools. One may think of using array extensions in synchronous languages, e.g., array iterators in Lustre and array of processes in Signal. This may not be completely satisfactory, as the useful element-wise semantics of arrays would be lost, leading, e.g., to a very conservative analysis of causality and single assignment. However, it is worth mentioning the recent works by Halbwachs and his colleagues on array properties in Lustre [128] [201]. The former work focuses on the analysis of array contents while the latter is dedicated to the synthesis of invariant properties in programs manipulating arrays. They certainly open new opportunities for considering synchronous languages as a support to analyze RSM specifications.

Furthermore, the loop transformation-based abstraction we proposed is another promising way to overcome the scalability issues related to the translation of the massive parallelism of RSM into synchronous languages.

FINAL OPINION ON THE PRESENTED CONTRIBUTIONS. The results exposed in this chapter contribute to a useful bridge between two modeling paradigms: repetitive structure modeling and synchronous reactive modeling. While they provided insightful observations on a possible exploitation of the complementary capabilities of these paradigms, their impact in industry is currently limited due to the absence of a complete and effective programming model infrastructure (e.g., combining their associated static analysis and loop transformation techniques) that facilitates their applicability. On the other hand, the two considered paradigms are borrowed by the

OMG Marte standard profile, which targets a wide audience of embedded system designers. So, I expect my contributions will help Marte users in a disciplined system design combining the related concepts.

An important question deserving attention that has not been answered in my contributions concerns a tight integration of loop transformations, array manipulation and clock calculus for a more efficient compilation of reactive data-intensive applications (see perspectives in Chapter 5).

Executive summary

MAIN COLLABORATIONS

- Sardes group (Inria/LIG, Grenoble)
- Aoste group (Inria/I3S, Sophia Antipolis)
- Computer & Embedded Systems (CES) Lab at ENIS (Sfax, Tunisia)

PROJECTS

- French Triade collaborative research action of Inria (other partners: Aoste and Espresso, 2008–2010)
- French ID-TLM initiative (other partners: Aoste and ST Microelectonics, 2008–2011)
- Franco-Tunisian Ksours project (other partner: CES-ENIS, Sfax, 2007 – 2011)

ADVISORY

- Huafeng Yu (PhD thesis from October 2005 to December 2008, 33%)
- Asma Charfi (Master thesis for eight months between April 2007 and June 2008, 50%)
- Mohamed Fellahi (Post-doc from October 2010 to February 2011, 50%)

SELECTED PUBLICATIONS

- *Conference*: IEEE International Conference on Embedded Systems and Software (ICESS), 2009 [104]
- *Journal*: ACM Transactions on Embedded Computing Systems (TECS), 2011 [106]
- *Journal*: Inderscience International Journal of Embedded Systems (IJES); 2010 [209]
- *Journal*: EURASIP Journal on Embedded Systems (EJES), 2008 [102]

CONTRIBUTION TO SOFTWARE: Gaspard2 (`http://www.gaspard2.org`)

4 DESIGN SPACE EXPLORATION FOR MPSOC CODESIGN

Contents

This chapter presents the last part of my contributions. The related studies, started from September 2008, concern the investigation of design space exploration (DSE) techniques for MPSoCs in Gaspard2 (see Figure 11). The basic principle is to rely on the capabilities of the two design paradigms considered in the previous chapters: on the one hand, the synchronous reactive approach and on the other hand, the repetitive structure modeling. These works have been conducted in the contexts of the PhD thesis of Adolf Abdallah, defended in March 2011 (co-advised with Jean-Luc Dekeyser), the ongoing PhD thesis of Xin An, started in October 2010 (co-advised with Éric Rutten), and the Post-doc fellowship of Rosilde Corvino, during one year from December 2009 (co-mentored with Pierre Boulet).

The chapter is organized as follows: in Section 4.1, I expose the motivations of the studies and I introduce the main challenges of interest; in Section 4.2, I address the design space exploration for efficient implementation of data-intensive applications specified in RSM on MPSoC architectures with optimized data transfer and storage; in Section 4.3, I present an abstract clock-based reasoning framework for the analysis and rapid prototyping of embedded applications executed on MPSoCs; finally, in Section 4.4, I discuss the strengths, limitations and future directions to the presented works. An executive summary is also given, regarding the key points of my contributions highlighted in this chapter.

4.1 OVERVIEW OF MAIN CHALLENGES

Data-intensive applications require high computing performance and parallelism as provided by well-adapted systems such as MPSoCs. In particular,

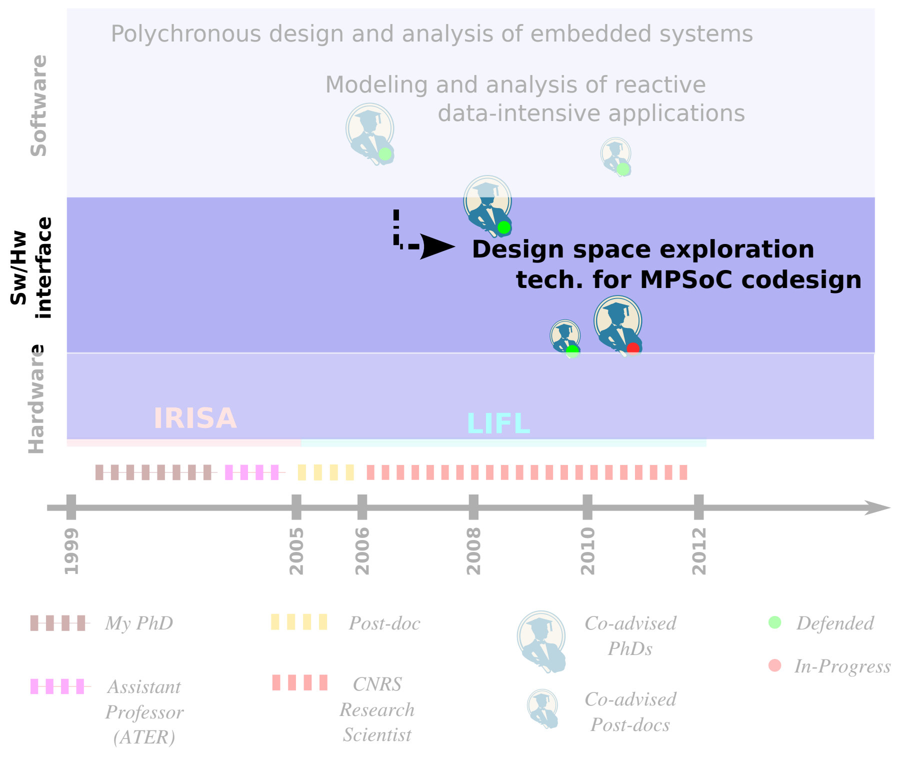

Software
Sw/Hw interface
Hardware
Polychronous design and analysis of embedded systems
Modeling and analysis of reactive data-intensive applications
Design space exploration tech. for MPSoC codesign
IRISA
LIFL
1999
2005
2006
2008
2010
2012
My PhD
Post-doc
Co-advised PhDs
Defended
In-Progress
Assistant Professor (ATER)
CNRS Research Scientist
Co-advised Post-docs

Figure 29: Specific contributions presented in the current chapter (the other contributions not exposed here are intentionally blurred).

for their implementation on MPSoCs, two factors have a strong impact on the quality of design results: on the one hand, the data transfer and storage micro-architecture [197] including the communication structure and the memory hierarchy, and on the other hand, the parallelism level of the computing resources. The first factor must be orchestrated so as to guarantee an optimized and bottleneck-free distribution of data to the different computing resources. The second factor is crucial in order to find a correct and energy-efficient software/hardware mapping. These two factors make the implementation of data-intensive applications on MPSoCs extremely complex. In this chapter, I advocate analysis methods that allow one to easily focus on relevant design issues in orthogonal ways.

4.1.1 *Data transfer and storage for efficient communications*

Many works deal with the design of MPSoCs for data-intensive applications at different abstraction levels: 1) the system level, where abstract analyses target the communication and storage mechanisms syntheses; 2) the processor level, where techniques such as High Level Synthesis (HLS), are aimed at rapidly inferring an efficient parallel Register Transfer Level model from a high level sequential specification. At these abstraction levels, loop-based methods have been largely used to explore design possibilities for data-intensive applications. At system level, they are used to estimate the storage requirements [137]. At processor level, loop-based HLS tools for data-intensive applications enable an implementation efficacy comparable to traditional flows, with the advantage of being automated [123].

Most of existing works improving HLS with loop transformations, optimize the loop iterations scheduling, reduce the redundant memory traffic and improve the synthesis of computation data path only for single nested loops. On the other hand, more abstract methods based on SDFs [166] are commonly used to explore communication structure and memory hierarchy for systems composed of multiple communicating loops. Unfortunately most SDF models do not take into account the multidimensionality of transferred data in data-intensive applications. Hence, they are not well-suited to describe the effects of loop transformations on multidimensional data-intensive application specification.

The first result presented in this chapter answers the following question:

First challenge.

How to explore communication-efficient implementations of data-intensive applications designed in RSM?

To answer the above question, I will exploit the loop transformations defined in RSM and explore the characteristics of the resulting application graph structures so as to isolate those with the most interesting data transfer and storage capacity. An evolutionary algorithm is used to accelerate the process.

4.1.2 *Software/hardware association for efficient execution*

Prototyping and simulation techniques have been considered as mainstream approaches to analyze MPSoCs design choices with respect to performance and energy-efficiency. Among these techniques, physical prototyping [19] involves circuit board and SoC in the form of working silicon. Emulation of hardware acceleration [135] involves field-programmable gate arrays (FPGAs) and require register transfer level (RTL) descriptions. While the major advantage of these two techniques is their high accuracy, they require a long time and provide a limited flexibility for an efficient DSE of multiple architectures.

Other approaches [5], [203] adopt the transactional level modeling (TLM) for fast simulation, and instruction set simulators (ISS) [19] for pre-silicon verification and debugging, by executing applications on hosts that simulate the processors of the target execution platform. However, the simulation speed and timing accuracy of ISS-based techniques are faster and less accurate than those of prototyping and emulation. Virtual system prototypes allowing cycle-accurate simulations are often preferred to these approaches. Further approaches rely on host-compilation [109], which uses back-annotations of timing estimates for a rapid yet accurate simulation. While the simulation speed is not affected by these notations, the accuracy of estimates quite depends on the ability to avoid possible pessimistic timing approximations obtained statically and unpredictable effects on timing approximations obtained dynamically. Trace-driven approach [134] is another solution found in literature, used for embedded system analysis.

In addition, we can also mention approaches focusing on static estimations of execution platform resources for applications with predictable behaviors, e.g., multimedia approaches. Typical reasoning models in these approaches are SDFs [247] [242].

From the above glance at prototyping and simulation techniques, we observe their complementarity regarding rapidity and accuracy for performance and energy estimation. About design correctness, these techniques consider debugging and testing, which are tedious and, with the ever increasing complexity of embedded systems, they are even a "nightmare" [75].

My proposition regarding the above concerns aims to give an answer to the following question:

Second challenge.

How to rapidly analyze, at system-design level, efficient mappings of applications on MPSoCs so as to find the best parallelism level for execution?

My suggested solution relies on an encoding of the problem with abstract clocks inspired by the multi-clock synchronous reactive modeling. This allows me to define a high-level modeling paradigm for combined software, hardware and environment specifications and to reason on it.

4.2 DSE FOR EFFICIENT DATA TRANSFER AND STORAGE

The present section summarizes my work on automatic exploration of communication-efficient architectures for the implementation of data-intensive applications, specified as graphs of nested loops in RSM (without inter-repetition dependencies in repetitive tasks). This work has been initiated in a close and fruitful collaboration with Rosilde Corvino since her Post-doc in the DaRT group (December 2009 – December 2010).

Rosilde Corvino is now research scientist and project manager at the Eindhoven Technical University (The Netherdands), and we still collaborate on the same topic.

4.2.1 *Related works*

Previous works on genetic algorithms [23] [174] have shown their efficacy in the optimization of multi-objective design explorations with large solution spaces. These works mostly construct the hardware architecture from a set of possible components, while we use a data-oriented configurable architecture with configuration parameters directly inferred from the application restructuring. As a result, the space of analyzed hardware solutions is narrowed to those appropriate for an analyzed application. Our method is similar to HLS-based flows applying loop transformations [168] [153] [200] [123] [137], but in contrast to these methods it targets applications including multiple communicating nested loops.

The use of loop transformations for architecture design and synthesis was already proposed in the 1990's [168] [153]. In [168], the authors use loop folding to schedule loop iterations in a pipeline fashion meeting scheduling and mapping constraints. Their work mostly focuses on the synthesis of the computing data paths and does not consider the synthesis of data transfer and storage micro-architectures. The approach [153] employs loop transformations for redundant memory traffic reduction, the optimal memory structure is neither explored nor selected.

Other recent works have used loop transformations for computing data path synthesis [200] [123] or memory architecture design [137]. As in our method, [200] uses loop transformations for FPGA design and defines abstract models of design area and performance to evaluate the effect of the loop transformations. But, this method targets computational data paths of single nested loops and the used transformations are oriented to instruction level parallelism, while our method mostly focuses on data parallelism.

In [123], the *Spark* framework uses code transformations such as code motion, dynamic renaming to improve the hardware implementation of computing resources and reduce the redundant memory traffic. This method is not aimed at exploring data parallelism nor the design of application-specific data transfer and storage micro-architectures. In [137], loop transformations are used to improve data transfer and storage micro-architecture by enhancing the data re-use.

4.2.2 *A hardware architecture template*

A generic hardware architecture model is considered consisting of a simplified representation of a tile-based MPSoC [24]. Each processing tile, *Proc Tile* i, contains a processing element T_i, local memories (light gray squares) and a local control for data access, *CTRL*. Thanks to a double-buffering mechanism, i.e., two local buffers alternatively read and written by the processing elements and the *CTRL* of a processing tile, data accesses and computations can be performed in parallel. Furthermore, a processing tile executes task repetitions in a pipeline due to the usage of pipelined computing units. Different tiles communicate through point-to-point links if they frequently exchange small amounts of data, or through a shared bus if they exchange large amounts of data.

Application/architecture mapping and scheduling rules make each RSM repetitive task (without inter-repetition dependencies) corresponds to a processing tile of the MPSoC and each tile of data to a local double-buffer. Figure 30 represents the customized architectures associated with the specification of Figure 17 in Chapter 3, which contains four repetitive tasks exchanging (large) arrays. As a consequence, four processing tiles are instantiated to execute the four tasks. The local memories of the instantiated processing tiles are able to store the data tiles of these tasks. They use a double-buffering mechanism to mask data access to the external memory performed through a shared bus.

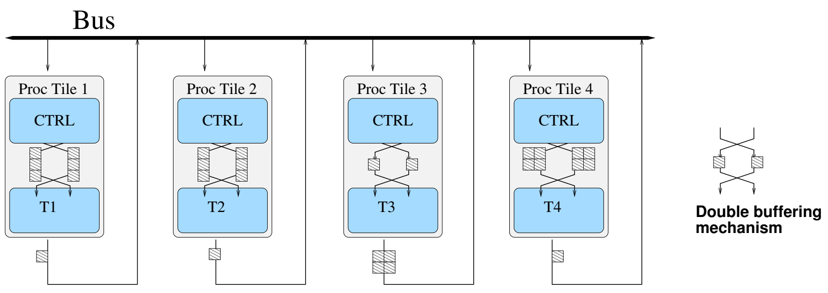


Figure 30: Architecture associated with the Array-OL model of Figure 17.

Loop transformations directly set specific mapping and scheduling rules on the considered tile-based hardware architecture, as follows:

1. The task *fusion* determines the communication structure. Indeed, when two tasks are merged they repeatedly exchange smaller data blocks. Thus, they are mapped onto a pipeline of processing tiles with point-to-point connections. They benefit from parallel read and write accesses to local double-buffers. By contrast, two unmerged tasks ex-

change larger data blocks that cannot be stored in local buffers. They are mapped onto processing tiles communicating via the shared bus with exclusive read/write accesses.

2. The *paving-change* determines different sizes of local double buffers.
3. The *tiling* determines different parallelism levels multiplying the number of processing elements within each processing tile.

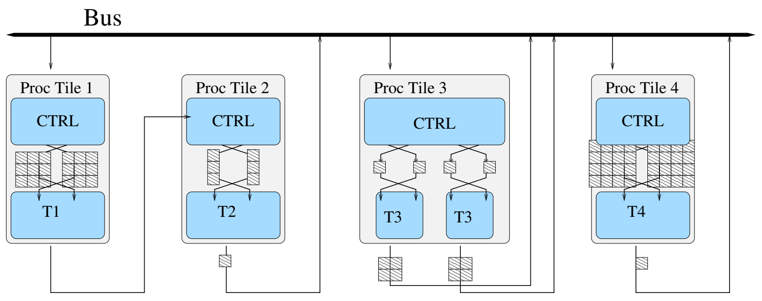


Figure 31: Architecture associated with the Array-OL model of Figure 18.

Figure 31 represents the customized architecture associated with the specification of Figure 18 in Chapter 3. The tasks T1 and T2 are merged by fusion. They are mapped onto processing tiles that communicate directly and realize a pipeline in our MPSoC architecture. *Proc Tile* 1 can copy data directly into the local memory of *Proc Tile* 2. The task T3 is tiled with two repetitions moved to the inner repetition level. Its corresponding processing tile implements two parallel processing elements with their own local buffers, that can process different data tiles in parallel. *Proc Tile* 3 uses a single shared controller to copy data into its local double-buffers in order to reduce chip area overhead due data parallelism increase.

4.2.3 *Overview of the DSE problem encoding*

Key publication for more details: [64].

AT A GLANCE. Our approach [64] exploits RSM and benefits from the associated loop transformations in order to systematically explore efficient implementation architectures, with respect to data transfer and storage. In Figure 32, as inputs, it takes an application specification in RSM and the previous customizable architecture template. The input application is transformed in order to enhance its data parallelism through data partitioning. In the same time, the architecture template is transformed exploring different application-specific customizations. The blocks of data manipulated by the application are streamed into the architecture. For the implementation of data-intensive computing systems, several parallelism levels are possible: inter-task parallelism realized through a systolic processing of data blocks; parallelism between data access and computation realized through the double-buffering mechanism; and data parallelism in a single task realized through a pipelining of the data stream processing or through the instantiation of parallel hardware resources.

The architecture parallelism level and the size of streamed data blocks are chosen in order to hide the latency of data transfers with computing time.

The above approach can be considered as a *meet-in-the middle* design technique because several hardware optimizations for data-intensive computing applications are already included into the customized architecture template.

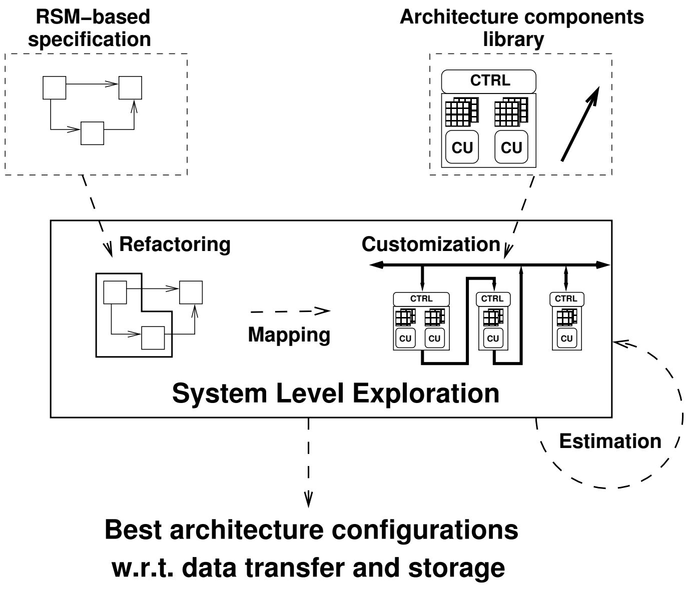


Figure 32: Overview of the proposed method.

The data transfer and storage configuration of this template is inferred from the analysis and refactoring of considered application specifications.

ENCODING OF THE SYSTEM LEVEL EXPLORATION PROBLEM. In [63], we considered a formalization using integer variables that represent required local buffer sizes and data parallelism respectively. In order to optimize design objectives, such as minimization of local buffers size and improvement of system temporal performance, some design constraints are formulated on these variables taking into account the chosen architecture/execution model. While an integer variable only expresses buffer size information here, the related DSE constraints involve additional aspects such as the time balancing between data access time and output computation time. So, our initial integer variable formalization was not expressive enough to fully capture all exploration aspects.

Key publications for more details: [63] and [64].

More recently, we proposed an alternative solution [62] relying on abstract clock encoding, which is more accurate than [63]. Thanks to this encoding, abstract clocks capture the order of data accesses, the time needed to read data and possible synchronizations between tasks. They also enable to characterize the way data consumption and production of repeated tasks are synchronized when they are executed on MPSoC processing tiles according to the mapping rules we defined. The loop transformations of RSM affect the data consumption and production rates of a task and, as a consequence, the associated abstract clocks. Furthermore, abstract clocks provide a uniform way to describe DSE constraints and objectives. This favors a simpler yet expressive formalization of the optimization problem, and thus opens the possibility to implement faster algorithms to solve design optimization problems.

Key publication for more details: [62].

4.2.4 *Implementation of our DSE approach*

The DSE approach implementation[1], shown in Figure 33, relies on the ab-

Key publication for further details [66].

stract clock encoding [66]. The inputs are RSM specification of applications and our customizable hardware architecture template. The outputs of the design exploration are Pareto pairs consisting of a restructured application and an optimized set of hardware architecture parameters, to which are associated abstract clocks. A few quality indicators about throughput, energy consumption and memory size are used to evaluate the design exploration outputs and guide the exploration itself toward the optimal solutions.

This implementation has been achieved in Java and involves three exploration steps [66], as follows:

Step 1: Fusion enumeration. This algorithm, equivalent to [211], enumerates all possible task fusions. It partitions them in a number of sub-spaces equal to the number of integer partitions of n, where n is the number of tasks in the application specification. An integer partition is a set of positive integer vectors whose components add up to n.

Step 2: Opt4J-based genetic algorithm. For each sub-space mapping a n integer partition, the Opt4J modular framework [173] supporting genetic algorithms, explores the tiling and paving change transformations in order to find the *local Pareto solutions*. All these solutions are merged in a new exploration space and passed to the final selection step.

Step 3: Final selection. This algorithm is an exhaustive search of the new formed exploration space in order to find *the global Pareto front*. An exhaustive exploration method is used for the fusion and a heuristic method for the other transformations because there is a finite and reasonably low number of possible fusions, while the number of possible tiling and paving-change is high.

4.2.5 *Some case studies*

We have demonstrated the validity of our approach by considering four sample applications: a JPEG encoder, a radar signal processing application named STAP [116], a hydrophone monitoring application named VBL [42] and a low pass spatial filter (LPSF). The exploration were performed on an Intel i7 quad-core processor running at 2.67 GHz with 4G of RAM.

RESULTS CHARACTERIZING THE EXPLORATION METHOD. Table 3 gives the complexity of the performed explorations, for each analyzed application. Typically, a JPEG encoder has 11 tasks and 75 possible fusions. The number of selected global Pareto solutions increases with the number of input tasks of an application, which characterizes the complexity of the problem.

	JPEG	STAP	VBL	LPSF
number of tasks in application	11	7	6	4
number of possible fusions	75	54	54	8
explored individuals	10^4	7900	7900	3200
total individuals in explo. space	315×10^6	9×10^5	9×10^5	2×10^3
average num. of global Pareto sol.	11	7	2	1

Table 3: Exploration complexity and selectivity.

1 Implemented in a tool available at `http://www.es.ele.tue.nl/~rcorvino/tools.html`.

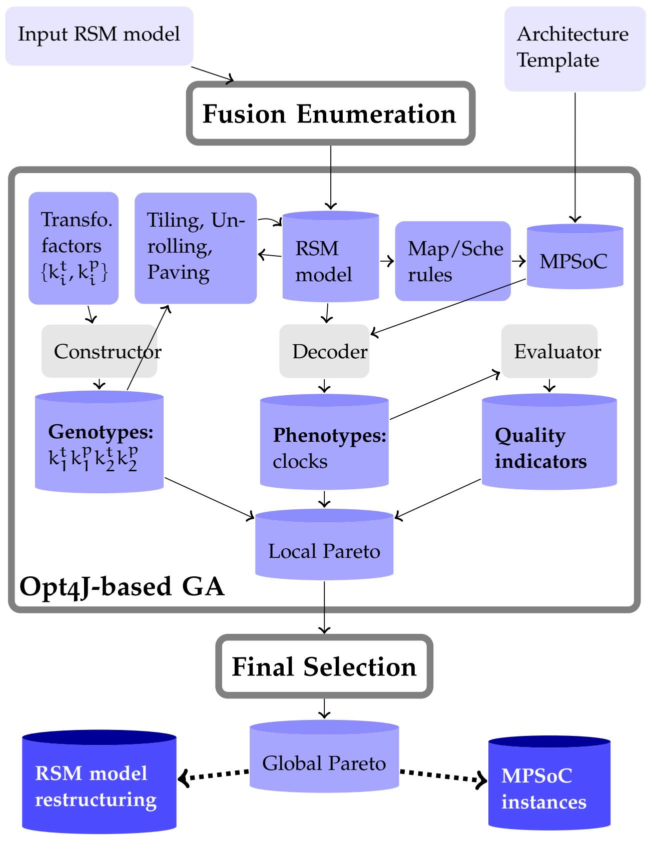


Figure 33: Implementation of our DSE approach.

Table 4 gives the run-time of the whole exploration for all applications. It shows the percentage of run-time for each exploration step: *fusion exploration*, *Opt4J-based genetic algorithm* and *final selection*. The latter step also includes the time spent to read and write text files. As expected, the run-time of the exploration depends on the size of the exploration space. Its major part is spent in the genetic algorithm by Opt4J. The percentage of time spent to perform the different steps is almost invariable with respect to the different applications and different trials per application. Furthermore, the total run-time is about seconds even for highly complex problem solving.

	JPEG	STAP	VBL	LPSF
run-time of exploration (sec.)	81	37	34	5
Perc. of run-time for Fusion Exploration	3%	3%	3%	2%
Perc. of run-time for Opt4J	96%	97%	97%	98%
Perc. of run-time for Final Selection	1%	$\approx$ 0%	$\approx$ 0%	$\approx$ 0%

Table 4: Exploration run-time.

To evaluate the precision of our approach, we compared it with an exhaustive search in Table 5. We give the run-time of the two exploration methods and the ϵ-indicators [248], asserting the quality of a Pareto front with respect to another. Here, $\epsilon = 1$ if the Pareto fronts obtained with both methods are identical. For feasibility reason, we performed the comparison for a gray-scale JPEG encoder (YJPEG in Table 5), and two reduced explorations of STAP and VBL applications (denoted STAP* and VBL* in Table 5).

	run-time		ϵ-indicator
	our method	exhaustive	
LPSF	5 sec.	14 min.	1
VBL	34 sec.	n.a.	-
STAP	37 sec.	n.a.	-
JPEG	81 sec.	n.a.	-
YJPEG	7 sec.	16 min.	1
STAP*	35 sec.	21 min.	1
VBL*	33 sec.	19 min.	1

Table 5: Quality of Pareto front search.

The above experiments show the relevance of our method in a design flow for an efficient and rapid exploration of data-intensive applications.

COMPARISON OF INFERRED SOLUTIONS W.R.T. THE STATE OF THE ART. We have assessed the quality of selected hardware architectures from our DSE results for the JPEG encoder against solutions existing in literature. We analyzed the results of two implementations by using a Virtex-4 XC4VFX20 FPGA[2] of Xilinx, one for a gray-scale JPEG and one for a color JPEG. We notice that since the solution considered from literature are implemented on older FPGA platforms (a Flex 10KE FPGA of Altera after hand-made optimizations and a Virtex-2 FPGA from a HLS tool) the comparison has to be considered as a rule of thumb indication of the possible improvements that can be obtained with our method.

Key publication for more details: [65].

We compared our implementation of the color JPEG encoder with [228], which is obtained with impulseC and is 100 times faster than a DSP-based implementation. The authors in [228] were able to process 41,000 blocks of 8x8 per second with a frequency of 50 MHz. We achieved a throughput of 312,500 blocks of 8x8 pixels per second for a Pareto solution, synthesized with the frequency of 50 MHz (maximum achievable frequency is 200 MHz). In our implementation, the local memories are implemented into dedicated optimized FPGA RAM's[3] for efficient data storage and rapid access to memory. Thus, their area occupancy is optimized. Our design implementation occupies 6% of slices and 88% of RAM's. The amount of used slices for logic is reasonably low and leaves room for implementing the processing elements. We have compared our implementation of a gray-scale JPEG encoder with the manual implementation of [6]. We achieve a throughput of 164 frames of 640×480 pixels compared to a throughput of 122 frames achieved with this implementation, for an area occupancy of 1% of slices and 8% of RAM's.

These experiments show that our implementations use a reasonably low logic area overhead and efficiently exploit the FPGA optimized RAM's to achieve a significant throughput increase.

4.2.6 *Benefits for MPSoC design frameworks*

The design space exploration technique presented above is a typical useful component of the toolkit required in high-level codesign frameworks to

2 We have reproduced these experiments with a more recent Virtex-6 FPGA. The obtained results are different but remain comparable with those observed with the Virtex-4 XC4VFX20 FPGA.
3 They are referred to as RAMB16's.

bring the crucial design decisions under control earlier. Gaspard2 is such a framework dedicated to data-intensive embedded systems. Even though our technique has not been fully integrated yet to Gaspard2, it can act as a companion tool to assist a user in the earlier design steps.

Given an application specified in RSM to be implemented on a multiprocessor hardware platform, our DSE solution is usable to automatically explore candidate application specification refactorings and hardware architecture configurations, which exhibit interesting properties regarding memory and communication concerns. The results would serve as insightful indications upon which the first design choices can rely. The best implementations achieved by our DSE can be automatically generated in the form of a graph by the associated tool, named DTSE[4] (Data Transfer and Storage Explorer). Such a graph illustrates the system structure corresponding to a given solution and allows a user to visualize and understand a selected design solution.

Our method can be also used as a complement to the approaches discussed in related works: it is possible to use our method to improve the architecture synthesis of a system with multiple loops, use pre-existing methods to improve the instruction level parallelism and, reduce redundant memory accesses inside the loop cores.

Finally, our technique is also under consideration in the ASAM[5] research project as part of the European ARTEMIS research program. The aim of ASAM is to automate the design of application-specific SoCs and processors using advanced DSE techniques. Rosilde Corvino, who is the main developer of our DSE tool, participates to ASAM. She focuses on the exploration of efficient application-specific instruction-set processors (ASIPs) designs. More generally, I believe our tool can be helpful for any system-level or processor-level design environment targeting data-intensive applications on MPSoCs.

4.3 A CLOCK MODEL FOR PERFORMANCE ANALYSIS IN MPSOCS

I present another high-level approach for a rapid assessment of MPSoC design [15] [93]. The concurrency of system behaviors is represented by abstract clocks inspired by the synchronous reactive approach. The way these behaviors are defined ensures by construction a correct system scheduling, w.r.t. specified data dependencies or event precedences, on multiprocessor execution platforms. The abstract clock modeling is flexible enough to address adaptive system behaviors, including changes of processor frequencies and task migration. The current study started during the PhD thesis of Adolf Abdallah (October 2007 – March 2011) and is now pursued in the PhD thesis of Xin An (started in October 2010).

The preliminary presentation of this approach [3] has been distinguished at SoC'2010 symposium (`http://soc.cs.tut.fi/2010/Best_paper_award.php`).

Adolf Abdallah is now Assistant Professor at Saint Joseph University in Beyrouth (Lebanon).

4.3.1 *Related works*

The static analysis of application designs with predictable behaviors, such as in the multimedia domain, has largely considered dataflow modeling, by using Kahn process networks (KPNs) and Synchronous dataflows (SDFs).

In [230], KPNs are used for design space exploration of multimedia applications on multiprocessor SoCs (MPSoCs). The authors do not investigate

4 `http://www.es.ele.tue.nl/~rcorvino/tools.html`.
5 ASAM: Automatic Architecture Synthesis and Application Mapping, `http://www.asam-project.org/`.

the design of scheduling algorithms for such applications. They rather consider them as a plug-in module which can be implemented as needed. As a result, simple scheduling schemes, like first come first serve algorithms, are considered in their experiments. In our framework, we study the admissible scheduling requirements, and propose a correct by construction scheduling algorithm.

A synchronous variant of KPNs [59], referred to as N-*synchronous KPNs* and based on periodic abstract clocks, has been defined with N-bounded channels for synchronizability analysis between processes. The finite value of the N parameter, corresponding to channels' size, is determined via a static analysis. Such an information can serve for memory dimensioning.

SDFs [166] capture the concurrent execution of applications and their analysis. Their authors developed a whole theory to statically schedule SDF graphs on homogeneous architectures. They proposed techniques for constructing *periodic admissible sequential* and *parallel schedules*, respectively referred to as PASS and PAPS. A period in PASS is constructed by computing the balance equations on data rates, while PAPS is achieved by constructing acyclic precedence graphs based on a number of periods of PASS. The proposed theory assumes homogeneity and uniform execution time on each processor, and not necessary synchronous processors. However, when scheduling an application on processors with different frequencies, it does not take into account the possible delays between processor clock cycles.

Another relevant scheduling algorithm is the self-time scheduling [221] in which a task is executed as soon as it is enabled, i.e., input data are ready. Therefore, its implementation requires specific execution platforms such as synchronous architectures. In [110], the authors propose an operational semantics for SDF graphs to analyze the throughput by describing self-timed executions in terms of labeled transition systems.

In [247], SDFs are used as in an optimization of streaming applications on heterogeneous execution platforms, mixing FPGA and CPUs. They are also used as representations in a design space exploration for multimedia application implementation [242]. SDFs are not explicitly clocked, which is a limitation for expressing multi-clock behaviors as required in combined software, hardware and environment specifications. For this reason in [246], authors consider a translation from SDF models to synchronous models.

In [237], the authors study the scheduling of real-time tasks on a heterogeneous platform with dynamic voltage and frequency scaling features. They propose a heuristic scheduling algorithm to explore the mapping choices from tasks to processor types and further frequencies with energy minimization as the goal. The algorithm considers independent tasks as the input, and thus requires neither to investigate the precedence relations, nor the potential delay between processor cycles.

Finally, in [196], authors address the design of multi-periodic embedded systems by considering the synchronous reactive approach. They study the monoprocessor scheduling of real-time tasks resulting from a translation of annotated synchronous dataflow programs. This study follows the usual schedulability analysis framework relying on the pioneer work of Liu and Layland [171]. It defines a code generation approach where given real-time constraints are satisfied. Our approach presented in the next section rather uses the abstract clock notion of synchronous dataflow languages to deal with schedulability concerns on multiprocessor platforms.

4.3.2 *Clock design of correct and efficient executions*

From this section, more details will be found in [15].

Our approach enriches the usual synchronous model [29] with quantitative time via abstract clocks. The resulting model provides a uniform support for design assessment w.r.t. quantitative properties beyond those addressed usually with the synchronous model.

In the sequel, I introduce a few basic definitions that are considered in our clock analysis system. We define models for application behavior, execution hardware platform and the mapping of both.

APPLICATION BEHAVIOR. We consider periodic embedded applications defined as a directed graph of tasks. These tasks exchange data according to the connections specified in an application graph. Each task has its own local activation clock according to which an associated sequence of events is observed. We construct our models by using the tagged signal system [167]. In the next, the following sets are assumed: a discrete set $\mathbb{T}$ of logical instants, having a smallest element τ_{min}, and associated with a partial order $\leqslant$; and a value domain $\mathbb{V}$.

Definition 1 (event) *An event e is a pair* (τ, ν)*, where* $\tau \in \mathbb{T}$ *is a logical instant, and* $\nu \in \mathbb{V}$ *is a value.*

The set $\mathcal{E}$ of all possible events is associated with *a partial order relation* $\prec$ such that: $\forall e_1 = (\tau_1, b_1), e_2 = (\tau_2, b_2), \tau_1 \leqslant \tau_2, \tau_1 \neq \tau_2 \Rightarrow e_1 \prec e_2$.

For a task, at most one event occurs at a logical instant. Such an event denotes the task is active at this instant. All events associated with the same task are totally ordered over the time. Given two events from different tasks, observed at the same logical instant, their respective precedence constraints w.r.t. all other events must be satisfied by each other. For instance, if events e_1 and e_2 occur at the same logical instant, then e_1 must satisfy all precedence constraints between e_2 and any other events, and *vice versa*.

Definition 2 (task and application behavior) *Given a task* t*, the behavior of* t*, denoted by* b_t*, is a totally ordered set of events. The behavior* b_T *of an application composed of a set* T *of tasks is a tuple* $(E, C, \prec)$ *where* E *is the set of events observed in all task behaviors, i.e.* $E = \bigcup b_{t, t \in T}$*,* C *is a precedence set composed of pairs of events* (e_i, e_j) *such that* $e_i \prec e_j$ *and* $\prec$ *is a precedence relation over* E*.*

Figure 34 illustrates an application behavior b_T with $T = \{t_0, t_1, t_2\}$, where $b_{t_0} = \{e_0^0, e_0^1\}, b_{t_1} = \{e_1^0, e_1^1\}$, $b_{t_2} = \{e_2^0, e_2^1\}$. Each event occurrence represents a task activation. The arrows represent precedences between events. For example, the arrow from event e_2^0 to event e_0^0 represents $e_0^0 \prec e_2^0$. The precedence set of this application behavior is $C = \{(e_0^0, e_2^0), (e_0^1, e_2^2), (e_0^1, e_1^2)\}$. The absence of arrow connection between two events means no precedence constraint between them, e.g., event e_0^1 and event e_1^0.

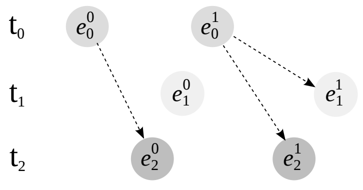


Figure 34: An application behavior b_T

The set of all possible application behaviors is denoted by $\mathcal{B}$. Many embedded applications have periodic behaviors. It is the case of streaming applications and time triggered applications. In our model, they are represented by a repetition of some application behavior patterns over the time.

Definition 3 (periodic application behavior) *Given a periodic application composed of a set of tasks* T*, its behavior* b_T *is defined as a pair* (π, ω)*, where* $\pi \in \mathcal{B}$ *is a behavior over* T*, repeated* ω *times over the time with* $\omega \in \mathbb{N}^*$*.*

EXECUTION PLATFORM BEHAVIOR. We consider an execution platform consisting of a set of processors operating synchronously according to a reference clock and communicating via a shared memory. The platform can be heterogeneous, meaning that different kinds of processing elements can be supported, e.g. processors, hardware accelerators, etc. However, the characteristics of all these processing elements are assumed to be known at design time, and particularly during mapping of applications on a hardware platform. They include the usual information provided in the data sheets of processing elements, e.g. range of possible values for frequencies w.r.t. the corresponding voltage levels.

Let P denote the set of processing elements in a platform. In our approach, we model platform behaviors through their clock activations according to given frequency values f_i for processing elements $p_i \in P, 1 \leqslant i \leqslant |P|$. We define the reference clock $\mathcal{K}$ of the platform with the frequency value calculated as $LCM(f_1, ..., f_{|P|})$, where LCM denotes the least common multiple. More concretely, the clock activations instants of the processing elements are modeled within a trace by considering the inverse of frequency values $1/f_i$, i.e. their period values. They are also referred to as processing element *clock cycles* in our approach.

We consider that a cycle $1/f_i$ of a processing element p_i is equal to a (integer) number of cycles $1/LCM(f_1, ..., f_{|P|})$ of the reference clock. We use $n_r(p_i)$ to denote this number. Figure 35 illustrates the behavior of a platform composed of three processors p_0, p_1 and p_2 with frequencies $f_1 = 100MHz, f_2 = 50MHz$ and $f_3 = 40MHz$.

	0	1	2	3	4	5	6	7	8	...
$\mathcal{K}$	●	●	●	●	●	●	●	●	●	...
p_0	●		●		●		●		●	...
p_1	●			●			●			...
p_2	●				●				●	...

Figure 35: Clock trace of processors

MAPPING OF APPLICATIONS ON EXECUTION PLATFORMS As in usual software/hardware codesign approaches, we define the mapping of applications on execution platforms. Such a decision usually precedes the scheduling of application tasks on processing elements.

Definition 4 (software-hardware mapping) *Given an application composed of a task set* T *and an execution platform formed of a processing element set* P*, a mapping of the application on the platform is defined as a total function* $M : T \rightarrow P$*.*

In the above definition, we notice that the inverse function $M^{-1} : P \rightarrow 2^{|T|}$ of mapping function M is not necessarily a total function since the processing elements of a platform may not be all used for execution.

After an application/execution platforms mapping, we associate each event e occurring in an application behavior b_T with two parameters $\alpha(e/p_i)$ and $\beta(e/p_i)$, respectively representing the number of processor cycles corresponding to its computation and communication costs w.r.t. considered processor elements p_i. These values are specified in terms of number of clock cycles and can be obtained statically by profiling their executions on the target processing elements.

SCHEDULING OF APPLICATIONS ON PLATFORMS. The scheduling decides when to execute the tasks specified in an application graph on selected processing elements of a platform from a mapping choice. Here, we assume that the events belonging to the same task behavior are always executed on the same processor, and the execution of a single event is non-preemptive.

In existing literature [239], four basic scheduling problems are distinguished: *i)* the unconstrained scheduling (UCS) consisting in finding a feasible (or optimal) schedule w.r.t. a set of operations O, a set of unit types U, a mapping function $m : O \rightarrow U$ and a partial order on O denoting precedence constraints; *ii)* the time-constrained scheduling (TCS) and *iii)* resource-constrained scheduling (RCS) problems, which respectively add time and resource constraints on UCS problem; and *iv)* the time- and resource-constrained scheduling (TRCS) problem, combining both TCS and RCS. To solve these problems, four scheduling techniques are discussed, namely as soon as possible/as late as possible ASAP/ALAP scheduling, list scheduling, force-directed scheduling and integer linear programming (ILP).

In our solution, we address an RCS problem with a list scheduling technique, which is a common choice for solving this problem [239]. We do not use ILP, which can also solve RCS problems, because of its inevitable cost to guarantee optimism and its unsuitability to deal with adaptive system behaviors. The scheduling algorithm defined in the following aims to define at which logical instants w.r.t. the reference clock, task events are executed on processing elements.

For convenience, we represent the schedules of tasks by means of a *ternary abstract clock* encoding. Such an abstract clock is a ternary-valued string over $\{-1, 0, 1\}$. The values 1 and 0 respectively represent the *active* and *idle* instants of a processing element executing some tasks w.r.t. the reference clock. The meaning of the value -1 is contextual: a sequence of -1 means active at these instants if it is preceded by 1, otherwise it denotes idle.

The scheduling of an application behavior $(E, C, \prec)$ consists of the scheduling of its elementary events E w.r.t. C. We first define the scheduling of an event. Based on it, we introduce three requirements regarding admissibility.

Definition 5 (schedule of an event) *A schedule of an event e on a processor p with parameters $\alpha(e/p)$, $\beta(e/p)$ and $n_r(p_i)$ is a ternary clock:*

$$clk(e/p)_{pos} = (1(-1)^{(\alpha(e/p)+\beta(e/p))*n_r(p)-1})_{pos}$$

where the subscript pos denote a reference position on the reference clock, indicating the start instant of e on p.

The schedule of an event encodes the beginning and the duration of the event execution, respectively denoted by pos and the length of its corresponding scheduling clock. In Figure 36, the schedule of event e_0^1 of task t_0 on p_0, where $\alpha + \beta = 1$ and $n_r(p_i) = 1$, is $(1(-1))_4$. This means from instant number 4 (according to reference clock), processor p_0 executes e_0^1 during the length of the clock $(1(-1))$.

Figure 36 shows three ternary clocks, representing the scheduling of the application behavior given in Figure 34 on the processors considered in Figure 35. For instance, from the ternary clock denoted by $\mathtt{clk}(t_0/p_0)$ (i.e., scheduling of task t_0 on processing element p_0), the execution of event e_0^0 starts from the very first instant of the reference clock, and takes one clock cycle of p_0. The execution of event e_0^1 starts at the fourth instant of the reference clock and takes one cycle. Between their executions, the clock has two idle instants.

In these ternary clocks, the value 1 indicates the logical instant at which the execution of the actions related to an event starts on the associated processor. The sequence of -1's following this value represents the duration of the whole event execution. The value 0 indicates the instant at which either an event has to wait for execution or task behaviors run to completion. Typically, a wait of event may happen upon i) synchronization w.r.t. precedence constraints, i.e., its preceding events have not finished, and ii) resource unavailability, i.e., its mapped processor is running another event.

	0	1	2	3	4	5	6	7	8	9
p_0	•		•		•		•		•	
$\mathtt{clk}(t_0/p_0)$	1	-1	0	-1	1	-1				
$\mathtt{clk}(t_1/p_0)$	0	1	-1	0	-1	-1	1	-1		
p_1	•			•			•			•
$\mathtt{clk}(t_2/p_1)$	1	-1	-1	0	-1	-1	1	-1	-1	

Figure 36: An example of task schedules in terms of ternary clocks

A nice feature of periodic ternary clocks is their compact representation. In Figure 36, the ternary clock $\mathtt{clk}(t_2/p_1)$ is written as $1(-1)^2 0(-1)^2 1(-1)^2$, where the exponent denotes the number of repetitions. Such a notation is quite adequate when manipulating periodic ternary clocks that capture the execution of periodic embedded applications on MPSoCs.

Definition 6 (correct schedule of a task behavior) *A correct schedule of a task behavior $b_t = (E, C, \prec)$ on a mapped processor p is a ternary clock $\mathtt{clk}(b_t/p)$, defined by the schedules of all events $e \in E$ such that the event precedence constraints defined by C are all satisfied.*

Typically, the schedule of a task behavior is constructed by synthesizing the schedules of all its events, i.e., by assembling their scheduling clocks properly according to their position $\mathtt{pos}$, together with waiting 0's in between if needed. Figure 37 gives three possible admissible schedules w.r.t. the example of Figure 36.

	0	1	2	3	4	5	6	7	8	9
p_0	•		•		•		•		•	
$\mathtt{clk}(t_0/p_0)$	1	-1	0	-1	1	-1				
$\mathtt{clk}(t_1/p_1)$	0	-1	1	-1	0	-1	-1	-1	1	-1
p_1	•			•			•			•
$\mathtt{clk}(t_2/p_1)$	0	-1	-1	1	-1	-1	1	-1	-1	

Figure 37: Admissible task schedules in terms of ternary clocks

Given the above definition, we define a correct schedule of an application behavior on a MPSoC platform as follows.

Definition 7 (correct schedule of application behavior) *A correct schedule of an application behavior* b_T *on an execution platform* P *w.r.t. a mapping* $M : T \to P$ *is a set of correct schedules of all tasks, i.e.* $\{clk(t/M(t)), t \in T\}$.

The schedule of an application behavior b_T on an MPSoC is represented as a set of ternary clocks, which correspond to the schedules of all its task behaviors. We have defined and implemented an algorithm that automatically generate correct application schedules on a multiprocessor execution platform, from given input specifications [15]. For periodic application behaviors, the resulting schedules are similar to those observed for regular static scheduling techniques applied in SDFs [35] or loop scheduling for software pipelining [234]. They may also serve to capture time-triggered application behaviors.

ABOUT ADAPTIVE BEHAVIORS. In the introductory chapter of this document, I pointed out the necessity for adaptivity in embedded systems, with respect to various factors: environment, execution platform, etc. Our clock-based reasoning framework has been extended accordingly to cope with adaptive system behaviors. To facilitate the generation of correct schedules in such cases, we propose a support to address typical adaptation requests like frequency variation and task migration. The ternary clock traces capturing a scheduling of an application can feature behaviors in which the processor allocations for tasks can change during execution. In the same way, behaviors featuring processor frequency changes during execution, e.g. for energy saving, can be described. In the current implementation for adaptive behaviors, the points in time where these changes occur are specified statically, before the scheduling.

4.3.3 *Performance analysis based on ternary clocks*

With the generated scheduling clocks of tasks and processors, various performance parameters can be analyzed in our framework.

The scheduling clocks of processors characterize the execution states of processors over the time. Given the scheduling clock $sclk(p_i)$ of a processor $p_i \in P$, it is quite direct to compute its execution time $ET(p_i)$ and usage ratio $UR(p_i)$ (useful for assessing the execution efficiency of p_i) as follows:

- $ET(p_i) = |sclk(p_i)| * \frac{1}{LCM(f_i, \ldots f_{|P|})}$
- $UR(p_i) = \frac{buzy\_cycles(sclk(p_i))}{buzy\_cycles(sclk(p_i)) + idle\_cycles(sclk(p_i))}$

where functions $buzy\_cycles(sclk(p_i))$ and $idle\_cycles(sclk(p_i))$ count the number of busy processor cycles and of idle processor cycles of a scheduling clock respectively.

The global execution time of an application behavior $A = (E, C, \prec)$, is the maximal execution time among all its mapped processors $p_i \in P$:

$$ET(A, P) = max\{ET(p_i), p_i \in P\}.$$

The other interesting performance parameter is the energy consumptions. Our framework is able to compute it, if provided with corresponding profiling results. For instance, given the energy consumption values for a busy cycle and for an idle cycle of a processor $p_i \in P$, denoted by $busy\_nrj(p_i)$ and

$\mathtt{idle\_nrj}(p_i)$ respectively, as well as its resulting scheduling clock $\mathtt{sclk}(p_i)$, we compute the energy consumptions $EC(p_i)$ of processor p_i as follows:

$$EC(p_i) = EC_{\mathtt{buzy}} + EC_{\mathtt{idle}}$$

where component $EC_{\mathtt{buzy}} = \mathtt{buzy\_cycles}(\mathtt{sclk}(p_i)) * \mathtt{busy\_nrj}(p_i)$ and component $EC_{\mathtt{idle}} = \mathtt{idle\_cycles}(\mathtt{sclk}(p_i)) * \mathtt{idle\_nrj}(p_i)$.

For a whole application $A = (E, C, \prec)$ executed on a set of processors $p_i \in P$, its global energy consumption is obtained as:

$$EPC(A, P) = \sum_{p_i \in P} EPC(p_i).$$

Beyond the above way to deal with energy, our clock model enables further reasoning possibilities about energy. For instance, we can consider the *slack time* of executed tasks, usually defined as the difference between their completion time and their associated deadline time. According to different frequency values of mapped processors, our ternary clock trace allows one to observe the variations of slack times. In particular, lower frequency values lead to longer completion times, hence shorter slack times. As a result, the configurations with shorter slack times are the best candidates for reduced[6] energy consumption. While this reasoning is somehow qualitative (in the sense that the energy consumption is not calculated to identify such configurations), the consumed energy EC can be estimated quantitatively as the product of execution time ET with power consumption information, obtained from profiling on given experimental platforms.

Finally, the scheduling clocks of tasks can be used to analyze the distance between the executions of two communicating events from different task behaviors within the same application period. Let us consider two communicating tasks A and B, where A produces a data block to feed B in each period. By computing the distance, and the number of produced events by A within this distance, we get indications about the required buffer size.

4.3.4 *Implementation of the clock-based framework*

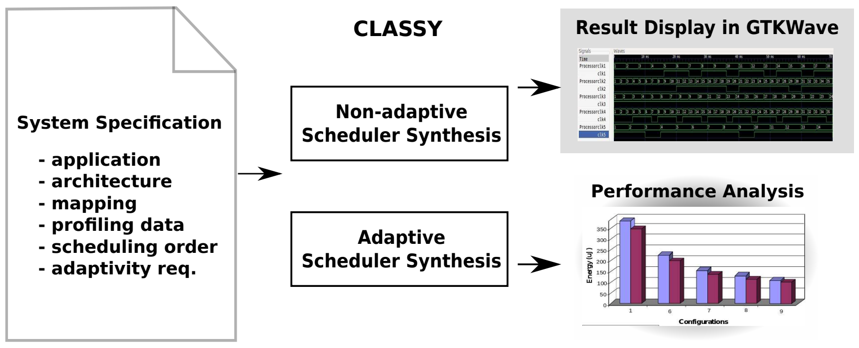


Figure 38: Overview the CLASSY tool.

Our prototype tool, named CLASSY (for CLock AnalySis SYstem), implements the modeling, scheduling and analysis approaches described in

6 We notice another way for energy minimizing, that consists in switching off processors temporarily when their assigned tasks are finished. In particular, it is interesting when the slack time is very long. However, switching on a processor has some cost in terms of delay and energy, which is necessary to bring the processor in a stable state for a new execution. This can increase the overall cost if it happens very frequently.

previous sections. It has around one thousand Java code lines and consists of the following modules as shown in Figure 38:

System specification provides interface for the user to define: application behavior (including task behaviors, precedence relation), execution platform behavior (including processors and frequencies), application-architecture mapping, performance analysis parameters like number of cycles evaluated for the computation of events and energy consumptions regarding the idle and active cycles of processors, and adaptation requirements (either frequency changing or task migration).

CLASSY which implements the heart of the tool by providing the ways to generate either application admissible schedules w.r.t. the system specification. The result comprises a set of scheduling clocks including the scheduling clocks of all tasks as well as the composed scheduling clocks of processors. It features either adaptive behaviors or not, according to the input specification requirements.

Performance analysis part of the tool computes, on the basis of generated scheduling clocks, the execution time, energy consumption as well as the distances between two events from different tasks. This last information is useful for estimating the waiting time between events, and for approximating possible buffer sizes between such events when there are some communications between their occurrences,

Result display in GTKWave generates a *vcd* file as an output to feed the standard simulation visualization tool GTKWave. This enables a user-friendly observation of resulting schedules of tasks on their mapped processors, as well as the running states of processors. By default, the analysis results are generated in textual form.

4.3.5 *A case study*

We applied our clock based approach and compared it with simulations in SoCLib [1]. Before going into details about the case study, I would like to mention that the simulations with SoCLib have been realized by Sarra Boumedien from ENIS-Sfax during her Master thesis in the DaRT group, from March 2011 to July 2011, under my supervison.

Another case study with the same approach can be found in [4]. It covers precedence correctness, temporal performance and energy consumption analysis.

SYSTEM SPECIFICATION. Let us consider a *motion JPEG (M-JPEG) decoding* algorithm as case study application. The M-JPEG decoder applies some filters to pixel streams for image decoding. The core algorithm manipulates 8×8-pixels blocks and is composed of five tasks as shown in Figure 39.

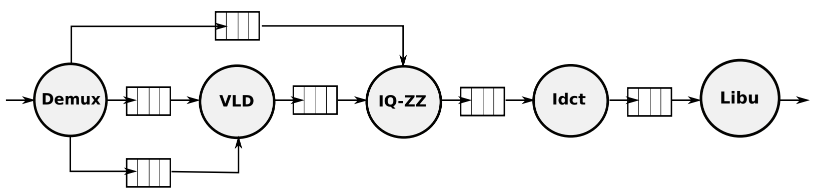


Figure 39: Application graph specifying the motion JPEG decoder

To execute the M-JPEG decoder, we consider a multiprocessor platform with a shared multi-bank memory, which can be configured to support up to five processors interconnected with a bus or a network-on-chip (NoC).

The studied mapping configurations are summarized in Table 6. We consider up to five processors $\{p_1, p_2, p_3, p_4, p_5\}$. For instance, in configuration number 1, all M-JPEG tasks are executed on processor p_1 while in configuration number 2, the successive tasks Demux, Vld, Iqzz are executed on p_1 and the successive tasks Idct and Libu are executed on p_2. In configuration number 3 also the same processors are considered, but the defined mapping does not select successive tasks to execute on the same processor. We refer to configurations like the number 2 as *successive task mappings* and configurations like the number 3 as *non successive task mappings*.

Configuration IDs	Mapping configurations ({tasks, mapped processors})
1	{{Demux, Vld, Iqzz, Idct, Libu}, p_1}
2	{{Demux, Vld, Iqzz},p_1},{{Idct, Libu},p_2}
3	{{Demux, Iqzz, Libu},p_1}, {{Vld, Idct}, p_2}
4	{{Demux, Vld}, p_1}, {Iqzz,p_2}, {{Idct, Libu}, p_3}
5	{{Demux, Iqzz}, p_1}, {{Vld, Libu}, p_2}, {Idct, p_3}
6	{Demux, p_1}, {Vld, p_2}, {Iqzz, p_3}, {{Idct, Libu}, p_4}
7	{{Demux, Libu},p_1}, {Vld, p_2}, {Iqzz, p_3}, {Idct, p_4}
8	{Demux, p_1}, {Vld, p_2}, {Iqzz, p_3}, {Idct,p_4}, {Libu,p_5}

Table 6: Analyzed mapping configurations

To achieve our experiments with CLASSY, we consider a periodic behavior of the decoder, illustrated in Figure 40. It is composed of two parts:

1. an initialization part, indicated by a blue curve, where some initial communications are achieved between the Demux task and the Vld and Iqzz tasks;
2. a periodic part, indicated by a red curve, which is repeated 36 times and consists of pixel block-wise decoding of an image.

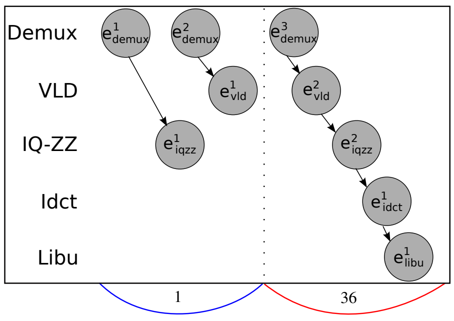


Figure 40: Application behavior for M-JPEG.

Table 7 gives the profiling data $\alpha(e) + \beta(e)$ corresponding to each event e shown in Figure 40. The given values are average values obtained from a profiling of the application in SoCLib.

COMPARISON OF SIMULATION RESULTS. A part of the simulation results obtained from our clock-based approach on the M-JPEG are reported in Figure 41, together with those observed with SoCLib. They represent the temporal performances associated with the mapping configurations summarized in Table 6. In Figures 41a and 41b all processors always operate at the

Tasks	Observed events	Num. of repetitions	Num. of processor cycles
Demux	$e^1_{\mathtt{demux}}$	1	12651
	$e^2_{\mathtt{demux}}$	1	21032
	$e^3_{\mathtt{demux}}$	36	2464
Vld	$e^1_{\mathtt{vld}}$	1	28042
	$e^2_{\mathtt{vld}}$	36	3007
Iqzz	$e^1_{\mathtt{iqzz}}$	1	1668
	$e^2_{\mathtt{iqzz}}$	36	4946
Idct	$e^1_{\mathtt{idct}}$	36	8978
Libu	$e^1_{\mathtt{libu}}$	36	1496

Table 7: Profiling data about M-JPEG tasks as inputs for CLASSY.

same frequency, while it is not the case in Figures 41c and 41d. Two system implementations are considered in SoCLib according to the communication infrastructure: bus *versus* NoC.

The experiments show that our clock-based approach yields results with similar tendency as those obtained with SoCLib. The precision of the results provided by CLASSY appears good when compared to the NoC-based results. However, it is not the case when considering the bus-based results. This observation is explained by the fact that NoCs offer higher communication performances than buses. The execution time obtained with NoCs is therefore shorter thanks to reduced communication time. In addition, possible bus access conflicts, which increase the communication overhead, lead to lower performances compared to NoC-based implementations. This issue is usually observed when the number of processors sharing the same bus gets higher. This may explain the increase of the execution time in Figure 41b, from configuration number 5 (three processors) to configuration number 8 (five processors). Since in our clock-based model of the M-JPEG application, the approximation of input profiling data given in Table 7 does not cover such communication overheads, the obtained results are less precise w.r.t. bus-based implementations.

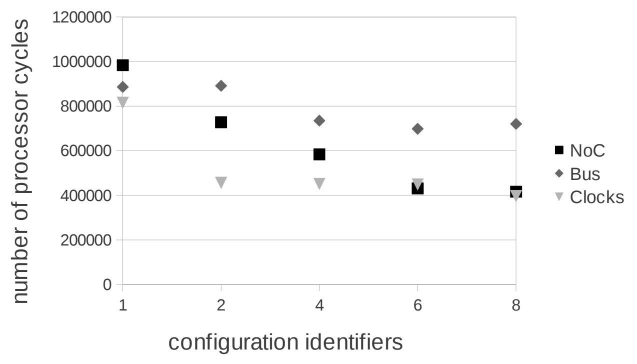
For the results obtained with processors operating at different frequencies in CLASSY, i.e. Figures 41c and 41d, the initialization part of the M-JPEG application behavior in Figure 40, has been scheduled first on the processor with the highest frequency.

On the other hand, to achieve the whole experiment (i.e., configure, execute and report), our approach requires a few minutes while SoClib necessitates several hours. Since it is faster and more flexible due to its high abstraction level, it must be considered for an early rapid exploration to reduce a design space, before applying simulation and prototyping.

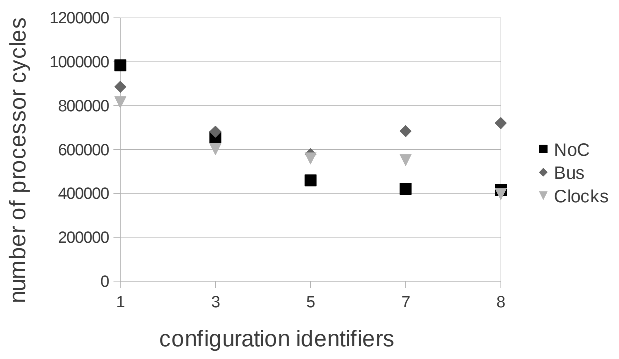
4.3.6 *Benefits of proposal for existing frameworks*

Our approach is a cost-effective and relevant means to facilitate the early analysis of design choices, before applying more advanced techniques, e.g. simulation and prototyping, which can be tedious are very expensive for complex designs. It does not aim to replace completely these techniques since its accuracy is limited due to the high abstraction; instead, it is an ideal complement to them. It is suitable for environments such as those adopting platform-based design [216], where high-level specifications of application

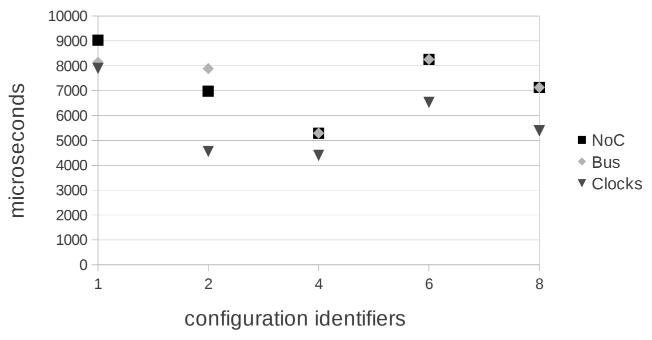
As a useful complement, the reader could refer to [93] for a more general clock-based approach addressing the design and analysis of streaming applications together with their environment and execution platforms.



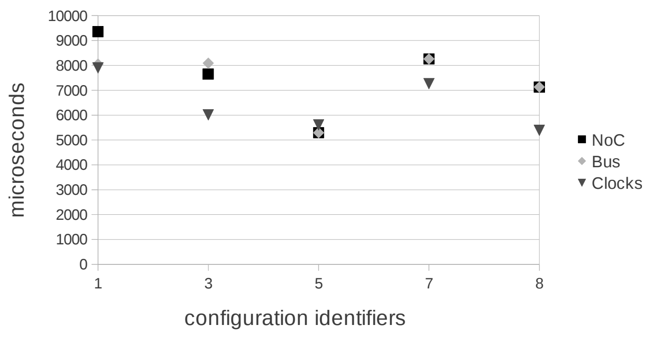
(a) Successive task mappings (proc. with same frequency).



(b) Non successive task mappings (proc. with same frequency).



(c) Successive task mappings (proc. without same frequency)



(d) Non successive task mappings (proc. without same frequency)

Figure 41: Execution times for M-JPEG decoder on an image: CLASSY *vs* SoCLib cycle-accurate simulations (comm. via bus and NoC).

functionality and hardware architecture are refined with well-characterized intellectual properties (IPs) and analyzed so as to rapidly converge towards design requirements.

The defined clock model can also play the role of a support for studying the parallelism level in data-intensive applications described in Gaspard2. Basically, the specification of an application in RSM expresses the potential (massive) parallelism inherent to algorithms. Since it is often the case that the actual embedded execution platforms do not offer all required processors for a full parallel execution, one has to deal with more constrained parallelism levels that fit well for an efficient execution. Our clock model can be used to tackle this issue by addressing various task concurrency scenarios via application behavior models and their associated schedules.

Finally, our clock model is a good candidate as an internal model for the analysis of specifications defined in, e.g. SDFs, synchronous languages or CCSL [16], in which clocks are manipulated as first-class citizens. The translation from SDFs can be defined based on periodic behavior traces corresponding to their self-time scheduling as illustrated in [246], while this is quite straightforward from the other formalisms since they manipulate very similar concepts.

4.4 SUMMARY AND DISCUSSION

In this chapter, an overview of my recent studies has been presented on design space exploration for MPSoCs in the Gaspard2 framework.

ON DSE FOR EFFICIENT DATA TRANSFER AND STORAGE. I exploited the complementarity of the capabilities offered by RSM and the synchronous reactive modeling. The former provides a useful way for application refactoring by applying its associated loop-like transformations to specifications. This allows various optimizations of a given application, for a range of target implementation platform. In this context, implementations with optimized data transfer and storage capabilities have been investigated. An MPSoC architecture template has been considered as a support for the realization of such implementations. I studied two possible encodings of the exploration problem by considering integer linear programming and an abstract clock formalization (inspired by synchronous languages). The latter was more expressive for reasoning on data accesses.

An important future research direction to this part of this contributions concerns the integration of further loop transformations in the presented method in order to explore more refactoring impacts on application implementations. Furthermore, the applicability of the proposed approach to hardware architecture exploration for application-specific instruction set processors is a promising perspective.

ON OUR CLOCK MODEL FOR PERFORMANCE ANALYSIS IN MPSOCS. Beyond the above study, which primarily targets communication concerns in MPSoCs, I have been developing a new design analysis framework that fully relies on abstract clocks. According to this framework, the executions of applications on multiprocessor platforms are encoded by clock traces, which are usable to assess to analyze both behavioral correctness and computation performance. The adaptivity of these executions has been taken into account. A preliminary comparison of this approach with an existing state-of-the-art framework shows promising results. The solution offered by this clock-based framework is to be considered as complementary of those obtained with simulations at low levels or physical prototyping. While it is less precise, it permits however to address design correctness and efficiency very rapidly at a negligible cost. This is an important advantage when dealing with the analysis of large and complex design spaces. In fact, the accuracy of approach quite depends on required input profiling information.

Future works include the integration of some heuristics in the proposed framework in order to make the design space exploration automatic. An example is to minimize energy consumption while still meeting a deadline. One can play with exploring the mappings or scheduling orders choices to achieve this. On the other hand, the clock-based analysis performed by CLASSY currently considers only a rough model of communications, which can lead to incoherent observations regarding an actual system execution, e.g. when resource access conflicts occur. So, one needs to investigate models of communication performances that reflect typical system architecture configurations. An important challenge is the approximation of the non deterministic behavior of communications, typically in presence of irregular memory access patterns when manipulating sparse matrices. The literature provides existing works that could be considered as possible inspiration. For instance in [48], authors propose a model for performance prediction and evaluation in point-to-point distributed communications for regular access patterns. In [208] [181], authors investigate an analysis technique in order to derive communication delay bounds and energy consumption in NoCs. While these techniques often adopt analytic models, specific methods like machine learning-based regression can be considered as in [143].

FINAL OPINION ON THE PRESENTED CONTRIBUTIONS. The results presented in this chapter are very recent and are still under development. They open a real opportunity to address the effective design space exploration for MPSoCs using RSM and synchronous reactive modeling paradigms. Their implementation within prototype tools, currently used in the European ASAM project by Rosilde Corvino and in the French ANR Famous project by Xin An, is a first promising step towards an adoption of the proposed techniques.

Executive summary

MAIN COLLABORATIONS

- Sardes group (Inria/LIG, Grenoble)
- Technical University of Eindhoven – TU/E (The Netherlands)

ADVISORY

- Adolf Abdallah (PhD thesis from October 2007 to March 2011, 50%)
- Xin An (**ongoing** PhD thesis from October 2010 to October 2013, 50%),
- Rosilde Covino (Post-doc from December 2009 to December 2010, 50%)
- Sarra Boumedien (Master thesis from March 2011 to July 2011, 100%)
- Mohamed-Hédi Ghaddab (Master thesis from February 2012 to June 2012, 100%)

PROJECT: French ANR Famous project (other partners: INRIA Rhône Alpes, Université de Bretagne Sud, Université de Bourgogne and Sodius, 2009–2013)

SELECTED PUBLICATIONS

- *Conference*: International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS), 2012 [66]
- *Workshop*: Workshop on Software and Compilers for Embedded Systems (SCOPES), 2012 [15]
- *Conference*: Design, Automation and Test in Europe (DATE), 2012 [93]
- *Bookchapter*: in Handbook of Data-Intensive Computing, Springer 2011 [64]

HONOR

- Best paper award for [3] at Symposium on System-on-Chip (SoC), 2010.

CONTRIBUTION TO SOFTWARE

- DTSE – Data Transfer and Storage Explorer (`http://www.es.ele.tue.nl/~rcorvino/tools.html`)
- CLASSY – CLock AnalysiS SYstem (available on-demand)

5 CONCLUSIONS AND PERSPECTIVES

Contents

The contributions presented in this document addressed the design of embedded systems, more generally implemented on distributed or multi-processor execution platforms, such as multiprocessor systems-on-chip (MP-SoCs). They considered the polychronous modeling (related to synchronous reactive approach), to specify and reason on the correctness of system concurrency. This choice has been motivated by all related advantages of such a modeling (expressivity, formal foundations, rich tool-set...), on which I have been strongly working for twelve years, in collaboration with colleagues at IRISA in Rennes (France) and Fermat Lab at Virginia Tech (VA, USA). Since I joined the DaRT group at LIFL in Lille (France) seven years ago, I have been developing a new design vision, which fosters the combination of the polychronous modeling with the repetitive structure modeling (RSM). RSM offers complementary design capabilities that are well-suited for parallelism expression at different levels in MPSoCs: application software, hardware architecture topology, hardware/software allocation.

5.1 OVERVIEW OF CONTRIBUTIONS

Based on the above two modeling paradigms, my contributions were organized into three chronological steps, corresponding to the three chapters following the introductory chapter in this document, as follows:

Chapter 2 presented the polychronous design of distributed embedded systems [95] [44] in Signal language [92], which covers my early works since my PhD. First, it discussed the definition of an adequate design methodology and its usage opportunity within the Signal design environment, Polychrony. This methodology included the definition of a library of various asynchronous mechanism models for communication, synchronization and execution, and the usage of specific program transformations to derive a correct distributed design of an application. Second, my studies on static analysis of combined logical/numerical clock properties in polychronous specifications [96] [146] have been reported. In particular, they concerned a usage of satisfiability modulo theory for the pragmatic reasoning on polychronous specifications has been exposed. Part of these works will keep on being investigated by the Espresso group at IRISA, in which I started my research, and the Fermat Lab at Virginia Tech.

Chapter 3 exposed a set of contributions obtained since my arrival in the DaRT group at LIFL. I have proposed a joint exploitation of both RSM and polychronous modeling paradigms to deal with the design of reactive data-intensive applications. The aim is to bridge the gap between data-parallel programming models manipulating multidimensional arrays and the synchronous dataflow programming model for an adequate design of target applications. In order to deal with time and dynamic behavior changes in RSM models of applications, an extension of RSM with reactive control modeling features [209] and its refinement towards synchronous dataflow languages [102] have been presented. Thanks to this refinement, relevant properties of RSM application designs become analyzable by using the rich verification tool-set dedicated to synchronous languages [104]. These contributions have been defined in the Gaspard2 codesign environment, dedicated to high-performance SoCs [106].

Chapter 4 presented my recent contributions focusing more on the efficient implementation and execution of data-intensive applications on MPSoCs. I have explored some hardware architecture properties by using high-level modeling, in order to identify the best implementation choices of an embedded system. This has been achieved again by considering the complementarity of RSM and polychronous modeling. A design space exploration (DSE) approach has been proposed to investigate implementations of applications, which provide optimized data transfer and storage capabilities [63] [64]. A complementary approach has been developed as a new design analysis framework according to which both behavioral correctness and computation performances can be dealt with [93] [4] [3]. It has been extended to support adaptive system execution. Both approaches aim to cover DSE concerns from communication and computation perspectives.

5.2 FUTURE RESEARCH TOPICS

My contributions strongly promote high-level modeling for the design and reasoning on systems. I believe high-level modeling offers a very relevant groundwork to address important features of future embedded systems, e.g., parallelism, adaptivity and heterogeneity. The approaches I have been studying are still deserving of more investigation, in a tight relation with hardware level details for a more accurate assessment of implementations. I have already drawn a certain number of short-term and in-the-medium-term research perspectives, in the summary and discussion sections of each chapter. Below, I give three major sub-topics that I would like to address in the next years. Figure 42 illustrates the central idea as a *continuum from high-level models to efficient implementations on adaptive MPSoCs*.

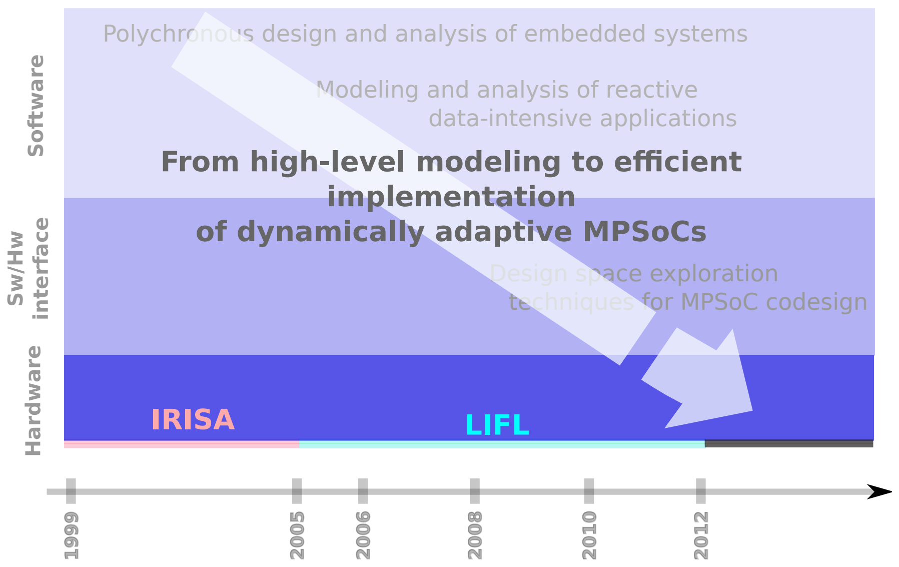

Polychronous design and analysis of embedded systems
Modeling and analysis of reactive data-intensive applications
From high-level modeling to efficient implementation of dynamically adaptive MPSoCs
Design space exploration techniques for MPSoC codesign
Software
Sw/Hw interface
Hardware
IRISA
LIFL
1999
2005
2006
2008
2010
2012

Figure 42: A schematic evolution of my research activities.

OVERVIEW. The first sub-topic about my future research activities concerns the synergistic exploitation of capabilities of synchronous dataflow languages and data-parallel languages for an efficient *codesign-aware compilation for MPSoCs*. It will largely benefit from the results already presented in this document. The second sub-topic refines the first one by dealing with the *safe management of MPSoC adaptivity*, which would rely on advanced technique such as discrete controller synthesis. The last sub-topic aims to gather fine-grain observations on system execution at lower abstraction levels, i.e., close to hardware architecture, for an *accurate observations for MPSoC adaptivity management*. There are several common points between these three research sub-topics and the challenges identified recently by the European Network of Excellence HiPEAC (High Performance and Embedded Architecture and Compilation) [77].

5.2.1 *Towards a codesign-aware compilation for MPSoCs*

The efficient execution of data-parallel applications with temporal constraints on adaptive MPSoCs requires compilation techniques which adequately exploit the characteristics of execution platforms, together with existing program analysis and transformations. Our works presented in Chapters 3 and 4 have sketched a bridge between the RSM formalism, which manipulates arrays and loops to describe regular data-intensive algorithms, and synchronous dataflow languages that adopt abstract clocks to describe reactive behaviors. These languages are associated with domain-specific compilation techniques that are deserving of reconciliation. On the one hand, the well-established polyhedral compilation applied, e.g., to RSM or the Alpha language [182] provides an efficient handling of multidimensional arrays and loop transformations [84] [85] [169]. It is well-suited for an efficient generation of nested loop code. On the other hand, the rich clock calculi implemented within compilers of synchronous dataflow languages, e.g. Signal [10] or Lucid Synchrone [37], allow a powerful analysis of control and synchronization issues in reactive programs. They permitted very interesting clock-driven compilation strategies that produce control flow-optimized sequential code.

A seamless combination of all these capabilities, where the provided analyses and optimizations are applied with respect to MPSoC codesign constraints will be new and extremely interesting. Typically, the potential loop transformations applied to data-parallel applications must be tightly aware of execution platform configurations. They can be driven by the target architectures of memory (e.g., multi-level caches or not) and processing elements (e.g., pipeline or not), as well as the performance characteristics of the architecture elements (e.g., CPU or hardware accelerator). Conversely, some function-specific analyses (e.g., communication or synchronization constraints regarding computations) may lead to application optimizations, which can have an impact on the way execution platform configurations are defined. This will open a real opportunity for aggressive optimizations of implementations for a wide range of performance-demanding MPSoCs, e.g., dedicated to multimedia applications.

The ideas illustrated in the analysis and DSE techniques presented in Chapter 4 may be considered in a new compilation framework for an efficient mapping of applications on MPSoC platforms. They would also serve to reason on both memory (data storage and transfer) and time requirements (related to synchronization of computation entities or events, real-time properties and temporal performances). There is to some extent a parallel between the vision advocated here and the observation of Lee about the need of time in computing, especially for a successful development of *cyber-physical systems*[1] [165]. Indeed, the usual abstraction levels considered for both programming and compilation should be made more physical resource-aware beyond the pure functional aspects, in order to fully exploit the characteristics of embedded systems.

Note that in many existing works dealing with similar analysis and optimization techniques, compilation concerns are mostly confined to application functionality [229], [128], [219], [185]. The originality of the research topic advocated in this section lies in that these concerns should be covered from the global MPSoC codesign viewpoint.

5.2.2 *Safe management of adaptivity in MPSoC codesign*

Adaptivity is a major ingredient to the reliability and performance of MPSoCs. In my previous contributions (see Chapters 3 and 4), I considered the management of system adaptivity by identifying the possible system behavioral modes before execution. Then, automata are used to describe the mode switching according to identified condition events. The advantage is that a typical verification technique like model-checking can be directly applied to check the correctness of expected control behavior.

However, with the increasing complexity of modern embedded systems, such an approach will become very tedious, if not infeasible. One reason

1 According to Wikipedia (`http://en.wikipedia.org/wiki/Cyber-physical_system`), "A cyber-physical system (CPS) is a system featuring a tight combination of, and coordination between, the system's computational and physical elements. Today, a precursor generation of cyber-physical systems can be found in areas as diverse as aerospace, automotive, chemical processes, civil infrastructure, energy, healthcare, manufacturing, transportation, entertainment, and consumer appliances. This generation is often referred to as *embedded systems*".

comes from a growing number of controllable and uncontrollable variables in systems, which leads to complex specification and verification of controller's properties that guarantee a correct adaptivity management. The recent advances in the connection of discrete controller synthesis (DCS) techniques to synchronous programming [40] [74] let me foresee an interesting opportunity to apply them in order to deal with this challenge. The advantage of DCS is that a reconfiguration controller can be automatically generated with existing tools [180] from a specification of expected control objectives. The synthesized controller is correct-by-construction, which saves long and tedious verification efforts.

DCS can be very useful for *virtualization*, which is a notable trend in MPSoCs implementation, where m virtual machines are hosted and executed on n physical computers, with $m > n$. The access to physical computer resources by virtual machines is under the control of a monitor, referred to as hypervisor [7]. This enables a flexible dynamic resource management, e.g., for load balancing or task migration. DCS could be considered for a suitable production of hypervisors meeting the requirements for an efficient resource management, as suggested in [32].

Recently, we have started a preliminary work that aims to investigate a similar question by considering DCS for the design of reconfigurable embedded systems. This work is part of a collaboration between Inria Lille, Inria Grenoble and *Université de Bretagne Sud*, within the framework of the French ANR Famous project to which I am participating. It covers a part of the ongoing PhD thesis of Xin An [14]. The insights expected from this study will certainly serve as a solid basis for me to tackle the issues mentioned above.

Accordingly, the compilation issues mentioned in the previous section should be considered more generally in such a dynamic management of system adaptivity, for better execution performances. In [231], a compiler support is used to enable an area-efficient implementation of a streaming application on dynamically reconfigurable processors. Further techniques such as *auto-tuning* [17] and *Just-in-Time* (JIT) compilers [157] [58], may be useful ingredients.

5.2.3 *Accurate observations for MPSoC adaptivity management*

To achieve an efficient codesign-aware compilation for adaptive MPSoCs, it is important to have a close look at lower layers in systems. As a matter of fact, this would provide fine-grain observations regarding the variation of performance metrics in a system, e.g., voltage, frequency, power. Thus, it is very important to take into account state-of-the-art techniques enabling us to get access to these information in a suitable and flexible manner.

Simulation techniques offer fine-grain observations about system behaviors, which are useful to address the design of computer system architectures. The results they provide may be considered to define a model of execution performance variations for a system, according to some identified configurations. SystemC has been used to define simulators at different abstraction levels, including the transaction-level modeling (TLM) [5] in the SoCLib environment [1]. In the same way, instruction set simulators (ISS's) [19] have been considered for the simulation of MPSoCs. The open-source and extensible Gem5 framework[2] provides a solution for computer system architecture design exploration by enabling the simulation of one or more

2 http://www.m5sim.org/wiki/index.php/Main_Page.

computer systems. I am currently supervising a Master thesis by Mohamed-Hédi Ghaddab (from ENIS – Sfax, from February 2012 to June 2012) on simulation experiments with this framework. A survey of similar existing simulators can be found in [192].

Beyond simulation, other similar techniques may also be considered, such as post-silicon measurements that help to accurately capture hardware variability [235], e.g., via the notion of hardware signatures [199]. Such a notion is defined by the post-manufacturing chip performance and power numbers crucial for the applications executed on an adaptive MPSoC. Typically, for multimedia applications, which often involve real-time and power consumption constraints, a signature can include the frequency deviations of hardware components and the leakage power dissipation values.

Thanks to the above two techniques, one may reasonably expect accurate input information to build some relevant static models of execution performance variations, usable by a controller to select configurations. Nevertheless, the accurate management of MPSoC adaptivity also calls for an on-line selection of correct and optimal execution configurations. Embedded systems often run under varying conditions, which are discovered only during execution time, e.g., evolution of power consumption, presence of resource contention, interaction events from environment. Thus, the occurrences of adaptivity events are often unpredictable. These events should be observed and treated on-line during executions. Note that hardware signatures can also be measured on-line during system execution at regular points in time. Then, they would better reflect the actual performance variations.

This part of my research perspectives is clearly a key step towards hardware-level design issues. Since I am not familiar enough with these issues, a good starting point is to consider my collaboration with colleagues having a strong background in microelectronics, such as the partners in the Famous ANR project, from *Université de Bretagne Sud and Université de Bourgogne*.

BIBLIOGRAPHY

[1] The SoClib Project, 2011. http://www.soclib.fr.

[2] Adolf Abdallah. *Conception de SoC à Base d'Horloges Abstraites : Vers l'Exploration d'Architectures en MARTE*. Ph.d. thesis, Université des Sciences et Technologie de Lille - Lille I, March 2011. URL http://tel.archives-ouvertes.fr/tel-00567963.

[3] Adolf Abdallah, Abdoulaye Gamatié, and Jean-Luc Dekeyser. Correct and Energy-Efficient Design of SoCs: the H.264 Encoder Case Study. In *International Symposium on System-on-Chip (SoC'2010)*, Tampere, Finland, 2010.

[4] Adolf Abdallah, Abdoulaye Gamatié, Rabie Ben Atitallah, and Jean-Luc Dekeyser. Abstract Clock-Based Design of a JPEG Encoder. *Embedded Systems Letters*, 0(0), 2012. Online at http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6158576.

[5] Samar Abdi, Yonghyun Hwang, Lochi Yu, Gunar Schirner, and Daniel D. Gajski. Automatic TLM Generation for Early Validation of Multicore Systems. *IEEE Design and Test of Computers*, 28:10–19, 2011.

[6] Luciano Volcan Agostini, Roger Carvalho Porto, Sergio Bampi, and Ivan Saraiva Silva. A FPGA based design of a multiplierless and fully pipelined JPEG compressor. In *Proc. of the 8th Euromicro Conference on Digital System Design (DSD'05)*.

[7] A. Aguiar and F. Hessel. Virtual hellfire hypervisor: Extending hellfire framework for embedded virtualization support. In *Quality Electronic Design (ISQED), 2011 12th International Symposium on*, pages 1 –8, march 2011.

[8] Luca Alfaro and Thomas A. Henzinger. Interface theories for Component-Based design. In Thomas A. Henzinger and Christoph M. Kirsch, editors, *Embedded Software*, volume 2211 of *Lecture Notes in Computer Science*, pages 148–165. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001. URL http://www.springerlink.com/content/0jqhuw40jlrbk8c7/.

[9] Eric Allen, David Chase, Joe Hallett, Victor Luchangco, Jan-Willem Maessn, Sukyoung Ryu, Guy L. Steele Jr., and Sam Tobin-Hochstadt. The Fortress Language Specification Version 1.0 Beta. Technical report, Sun Microsystems, Inc., March 2007.

[10] Pascalin Amagbégnon, Loïc Besnard, and Paul Le Guernic. Implementation of the data-flow synchronous language signal. In *Proceedings of the ACM SIGPLAN 1995 conference on Programming language design and implementation*, PLDI '95, pages 163–173. ACM, 1995.

[11] Abdelkader Amar, Pierre Boulet, and Philippe Dumont. Projection of the array-ol specification language onto the kahn process network computation model. In *8th International Symposium on Parallel Architectures, Algorithms, and Networks, ISPAN 2005, December 7-9. 2005, Las Vegas, Nevada, USA*, pages 496–503, 2005.

[12] Saman Amarasinghe, Dan Campbell, William Carlson, Andrew Chien, William Dally, Elmootazbellah Elnohazy, Mary Hall, Robert Harrison, William Harrod, Kerry Hill, Jon Hiller, Sherman Karp, Charles Koelbel, David Koester, Peter Kogge, John Levesque, Daniel Reed, Vivek Sarkar, Robert Schreiber, Mark Richards, Al Scarpelli, John Shalf, Allan Snavely, and Thomas Sterling. ExaScale Software Study: Software Challenges in Extreme Scale Systems. Technical report, 2009. URL http://users.ece.gatech.edu/mrichard/

ExascaleComputingStudyReports/ECSS%20report%20101909.pdf. Report, Vivek Sarkar, Editor & Study Lead.

[13] Saman P. Amarasinghe, Jennifer-Ann M. Anderson, Monica S. Lam, and Chau-Wen Tseng. An overview of the suif compiler for scalable parallel machines. In *PPSC*, pages 662–667, 1995.

[14] Xin An, Abdoulaye Gamatié, and Éric Rutten. Safe design of dynamically reconfigurable embedded systems. In *Workshop on Model Based Engineering for Embedded Systems Design (M-BED'2011)*. ECSI, 2011.

[15] Xin An, Sarra Boumedien, Abdoulaye Gamatié, and Éric Rutten. CLASSY: a Clock Analysis System for Rapid Prototyping of Embedded Applications on MPSoCs. In *Proceeding of the 15th International Workshop on Software and Compilers for Embedded Systems (SCOPES'2012), St. Goar, Germany*. ACM, May 2012.

[16] Charles André and Frédéric Mallet. Clock Constraints in UML/MARTE CCSL. Research Report RR-6540, INRIA, 2008. URL http://hal.inria.fr/inria-00280941/PDF/rr-6540.pdf.

[17] Jason Ansel, Yee Lok Won ans Cy Chan, Marek Olszewski, Alan Edelman, and Saman Amarasinghe. Language and compiler support for auto-tuning variable-accuracy algorithms. In *The International Symposium on Code Generation and Optimization*, Chamonix, France, Apr 2011. URL http://groups.csail.mit.edu/commit/papers/2011/ansel-cgo11-pbaccuracy.pdf.

[18] Pascal Aubry. *Mises en œuvre distribuées de programmes synchrones.* Ph.d. thesis, Université de Rennes I, IFSIC, France, October 1997.

[19] Brian Bailey and Grant Martin. *ESL Models and their Application: Electronic System Level Design and Verification in Practice*. Springer Publishing Company, Incorporated, 2010.

[20] Ana Balevic and Bart Kienhuis. A data parallel view on polyhedral process networks. In *Proceedings of the 14th International Workshop on Software and Compilers for Embedded Systems*, SCOPES '11, pages 38–47. ACM, 2011. ISBN 978-1-4503-0763-5.

[21] Utpal Banerjee, Rudolf Eigenmann, Alexandru Nicolau, and David A. Padua. Automatic program parallelization. *Proceedings of the IEEE*, 81(2):211 –243, feb 1993.

[22] Remi Barrere, Eric Lenormand, Dai Bui, Edward A. Lee, Christopher Shaver, and Stavros Tripakis. An introduction to the pthales domain of ptolemy ii. Technical Report UCB/EECS-2011-32, EECS Department, University of California, Berkeley, Apr 2011. URL http://www.eecs.berkeley.edu/Pubs/TechRpts/2011/EECS-2011-32.html.

[23] Twan Basten, Emiel Van Benthum, Marc Geilen, Martijn Hendriks, Fred Houben, Georgeta Igna, Frans Reckers, Sebastian De Smet, Lou Somers, Egbert Teeselink, Nikola Trčka, Frits Vaandrager, Jacques Verriet, Marc Voorhoeve, and Yang Yang. Model-driven design-space exploration for embedded systems: the Octopus toolset. In *Proceedings of the 4th international conference on Leveraging applications of formal methods, verification, and validation - Volume Part I*, ISoLA'10, pages 90–105. Springer-Verlag, 2010. ISBN 3-642-16557-5, 978-3-642-16557-3.

[24] Lucas Benini and Giovanni De Micheli. Networks on chips: a new soc paradigm. *Computer*, 35:70 –78, jan 2002.

[25] Albert Benveniste. Safety critical embedded systems: the Sacres approach. In *proceedings of Formal techniques in Real-Time and Fault Tolerant Systems, FTRTFT'98 school, Lyngby, Denmark*, September 1998.

[26] Albert Benveniste, Paul Le Guernic, and Christian Jacquemot. Synchronous programming with events and relations: the Signal language and its semantics. *Sci. Comput. Program.*, 16(2):103–149, 1991.

[27] Albert Benveniste, Benoît Caillaud, and Paul Le Guernic. From synchrony to asynchrony. In *International Conference on Concurrency Theory*, pages 162–177, 1999.

[28] Albert Benveniste, Paul Caspi, Paul Le Guernic, Hervé Marchand, Jean-Pierre Talpin, and Stavros Tripakis. A protocol for loosely time-triggered architectures. In *Conference on Embedded Software (EMSOFT'02), J. Sifakis and A. Sangiovanni-Vincentelli, Eds, LNCS vol 2491, Springer Verlag*, 2002.

[29] Albert Benveniste, Paul Caspi, Stephen A. Edwards, Nicolas Halbwachs, Paul Le Guernic, and Robert De Simone. The synchronous languages twelve years later. *Proceedings of the IEEE*, 91(1):64–83, January 2003.

[30] Albert Benveniste, Timothy Bourke, Benoît Caillaud, and Marc Pouzet. A hybrid synchronous language with hierarchical automata: static typing and translation to synchronous code. In *Proceedings of the ninth ACM international conference on Embedded software (EMSOFT'11)*, pages 137–148. ACM, 2011.

[31] Gérard Berry and Ellen Sentovich. An implementation of constructive synchronous programs in POLIS. *Formal Methods in System Design*, 17(2):135–161, 2000.

[32] Nicolas Berthier, Florence Maraninchi, and Laurent Mounier. Synchronous Programming of Device Drivers for Global Resource Control in Embedded Operating Systems. In *Proceedings of the 2011 SIGPLAN/SIGBED Conference on Languages, Compilers and Tools for Embedded Systems*, LCTES '11, pages 81–90, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0555-6. doi: http://doi.acm.org/10.1145/1967677.1967689.

[33] Loïc Besnard, Thierry Gautier, and Paul Le Guernic. Signal V4 – INRIA version: Reference Manual, 2010. www.irisa.fr/espresso/Polychrony/document/V4_def.pdf.

[34] Frédéric Besson, Thomas Jensen, and Jean-Pierre Talpin. Polyhedral analysis for synchronous languages. In *Proceedings of the 6th International Symposium on Static Analysis, volume 1694 of Lecture Notes in Computer Science*, pages 51–68. Springer-Verlag, September 1999.

[35] Shuvra S. Bhattacharyya. *Compiling Dataflow Programs for Digital Signal Processing*. Ph.d. thesis, EECS Department, University of California, Berkeley, 1994. URL http://www.eecs.berkeley.edu/Pubs/TechRpts/1994/2589.html.

[36] Shuvra S. Bhattacharyya, Gordon Brebner, Jörn W. Janneck, Johan Eker, Carl von Platen, Marco Mattavelli, and Mickaël Raulet. Opendf: a dataflow toolset for reconfigurable hardware and multicore systems. *SIGARCH Comput. Archit. News*, 36:29–35, June 2009.

[37] Dariusz Biernacki, Jean-Louis Colaço, Gregoire Hamon, and Marc Pouzet. Clock-directed modular code generation for synchronous data-flow languages. In *Proceedings of the 2008 ACM SIGPLAN-SIGBED conference on Languages, compilers, and tools for embedded systems*, LCTES '08, pages 121–130. ACM, 2008.

[38] Guy E. Blelloch. Nesl: A nested data-parallel language (version 2.6). Technical report, Pittsburgh, PA, USA, 1993.

[39] Shekhar Y. Borkar, Hans Mulder, Pradeep Dubey, Stephen S. Pawlowski, Kevin C. Kahn, Justin R. Rattner, and David J. Kuck. Platform 2015: Intel processor and platform evolution for the next decade. Technical report, Intel, 2005. URL http://epic.hpi.uni-potsdam.de/pub/Home/TrendsAndConceptsII2010/HW_Trends_borkar_2015.pdf. White paper.

[40] Tayeb Bouhadiba, Quentin Sabah, Gwenaël Delaval, and Éric Rutten. Synchronous control of reconfiguration in fractal component-based systems: a case study. In Samarjit Chakraborty, Ahmed Jerraya, Sanjoy K. Baruah, and Sebastian Fischmeister, editors, *EMSOFT*, pages 309–318. ACM, 2011. ISBN 978-1-4503-0714-7.

[41] Pierre Boulet. Formal Semantics of Array-OL, a Domain Specific Language for Intensive Multidimensional Signal Processing. Technical report, INRIA, France, March 2008. available online at http://hal.inria.fr/inria-00261178/fr.

[42] Pierre Boulet, Jean-Luc Dekeyser, Jean-Luc Levaire, Philippe Marquet, Julien Soula, and Alain Demeure. Visual data-parallel programming for signal processing applications. In *Proc. of Ninth Euromicro Workshop on Parallel and Distributed Processing, 2001*.

[43] Pierre Boulet, Alain Darte, Georges-André Silber, and Frédéric Vivien. Loop parallelization algorithms: From parallelism extraction to code generation. *Parallel Computing*, 24(3-4):421–444, 1998. URL http://hal.inria.fr/inria-00565000/en/.

[44] Christian Brunette, Jean-Pierre Talpin, Abdoulaye Gamatié, and Thierry Gautier. A metamodel for the design of polychronous systems. *Journal of Logic and Algebraic Programming*, 78(4):233 – 259, 2009.

[45] Randal E. Bryant. Symbolic manipulation of boolean functions using a graphical representation. In *DAC*, pages 688–694, 1985.

[46] Darius Buntinas, Guillaume Mercier, and William Gropp. Implementation and evaluation of shared-memory communication and synchronization operations in mpich2 using the nemesis communication subsystem. *Parallel Comput.*, 33: 634–644, September 2007.

[47] Surendra Byna and Xian-He Sun. Special issue on data-intensive computing. *Journal of Parallel and Distributed Computing*, 71(2):143 – 144, 2011.

[48] Kirk W. Cameron and Rong Ge. Predicting and evaluating distributed communication performance. In *Proceedings of the 2004 ACM/IEEE conference on Supercomputing (SC'04)*, pages 43–. IEEE Computer Society, 2004. ISBN 0-7695-2153-3.

[49] William W. Carlson, Jesse M. Draper, David Culler, Kathy Yelick, Eugene Brooks, and Karren Warren. Introduction to UPC and Language Specification. Technical Report CCS-TR-99-157, Bowie, MD, May 1999.

[50] Paul Caspi and Marc Pouzet. Synchronous Kahn networks. In *International Conference on Functional Programming*, pages 226–238, 1996.

[51] Paul Caspi, Adrian Curic, Aude Maignan, Christos Sofronis, Stavros Tripakis, and Peter Niebert. From simulink to scade/lustre to tta: a layered approach for distributed embedded applications. In *Proceedings of the 2003 ACM SIGPLAN conference on Language, compiler, and tool for embedded systems*, LCTES '03, pages 153–162, New York, NY, USA, 2003. ACM. ISBN 1-58113-647-1.

[52] Bradford L. Chamberlain, David Callahan, and Hans P. Zima. Parallel programmability and the chapel language. *Int. J. High Perform. Comput. Appl.*, 21: 291–312, August 2007.

[53] Daniel Marcos Chapiro. *Globally Asynchronous Locally Synchronous Systems*. Ph.d. thesis, Stanford University, 1984.

[54] Asma Charfi, Abdoulaye Gamatié, Antoine Honoré, Jean-Luc Dekeyser, and Mohamed Abid. Validation de modèles dans un cadre d'IDM dédié à la conception de systèmes sur puces. In *4èmes Journées sur l'Ingéniérie Dirigée par les Modèles – IDM'08*, 2008.

[55] Philippe Charles, Christian Grothoff, Vijay Saraswat, Christopher Donawa, Allan Kielstra, Kemal Ebcioglu, Christoph von Praun, and Vivek Sarkar. X10: an object-oriented approach to non-uniform cluster computing. In *Proceedings of the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, OOPSLA '05, pages 519–538, New York, NY, USA, 2005. ACM. ISBN 1-59593-031-0.

[56] Bruno Chéron. *Transformations syntaxiques de Programmes Signal*. Ph.d. thesis, Université de Rennes I, IFSIC, France, September 1991.

[57] Anthony Coadou. *Réseaux de processus flots de données avec routage pour la modélisation de systèmes embarqués*. Ph.d. thesis, Université de Nice Sophia-Antipolis, December 2010. URL http://tel.archives-ouvertes.fr/tel-00545008.

[58] Albert Cohen and Erven Rohou. Processor virtualization and split compilation for heterogeneous multicore embedded systems. In *DAC*, pages 102–107, 2010.

[59] Albert Cohen, Marc Duranton, Christine Eisenbeis, Claire Pagetti, Florence Plateau, and Marc Pouzet. N-synchronous Kahn networks. In *ACM Symp. on Principles of Programming Languages (PoPL'06)*, Charleston, South Carolina, USA, January 2006.

[60] Jean-Louis Colaço, Grégoire Hamon, and Marc Pouzet. Mixing signals and modes in synchronous data-flow systems. In *EMSOFT*, pages 73–82, 2006.

[61] F. Commoner, Anatol Holt, Shimon Even, and Amir Pnueli. Marked directed graphs. *J. Comput. Syst. Sci.*, 5(5):511–523, October 1971. ISSN 0022-0000. doi: 10.1016/S0022-0000(71)80013-2.

[62] Rosilde Corvino and Abdoulaye Gamatié. Abstract Clocks for the DSE of Data-Intensive Applications on MPSoCs. In *Proceeding of the 4th IEEE International Workshop on Multicore and Multithreaded Architectures and Algorithms (M2A2 2012), Leganés, Madrid*. IEEE, July 2012.

[63] Rosilde Corvino, Abdoulaye Gamatié, and Pierre Boulet. Architecture exploration for efficient data transfer and storage in data-parallel applications. In *Euro-Par (1)*, pages 101–116, 2010.

[64] Rosilde Corvino, Abdoulaye Gamatié, and Pierre Boulet. Design Space Exploration for Efficient Data-Intensive Computing on SoCs. In Borko Furht and Armando Escalante, editors, *Handbook of Data-Intensive Computing*. Springer, 2011.

[65] Rosilde Corvino, Erkan Diken, Abdoulaye Gamatié, and Lech Jozwiak. Transformation-based Exploration of Data-Parallel Architecture for Customizable Hardware: A JPEG Encoder Case Study. In *Euromicro Conference on Digital System Design (DSD 2012)*, Cesme, Izmir, Turkey, September 2012. IEEE.

[66] Rosilde Corvino, Abdoulaye Gamatié, Marc Geilen, and Lech Jozwiak. Design Space Exploration in Application-Specific Hardware Synthesis for Multiple Communicating Nested Loops. In *International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS XII)*, Samos, Greece, July 2012. IEEE.

[67] Georges Coulouris, Jean Dollimore, and Tim Kindberg. *Distributed systems (4th ed.): concepts and design*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2005.

[68] Ron Cytron, Jeanne Ferrante, Barry K. Rosen, Mark N. Wegman, and F. Kenneth Zadeck. Efficiently computing static single assignment form and the control dependence graph. *ACM Trans. Program. Lang. Syst.*, 13:451–490, October 1991.

[69] Leonardo de Moura and Nikolaj Bjorner. Satisfiability Modulo Theories: An Appetizer. In *Brazilian Symposium on Formal Methods (SBMF'2009), Gramado, Brazil*, August 2009.

[70] Jean-Luc Dekeyser, Abdoulaye Gamatié, Samy Meftali, and Imran Rafiq Quadri. Models for Co-Design of Heterogeneous Dynamically Reconfigurable SoCs. In Nicolescu, Gabriela; O'Connor, Ian; Piguet, and Christian, editors, *Heterogeneous Embedded Systems - Design Theory and Practice*, page 26 p. Springer, 2012. URL http://hal.inria.fr/inria-00525023/en.

[71] J.L. Dekeyser, A. Gamatié, A. Etien, R.B. Atitallah, and P. Boulet. Using the uml profile for marte to mpsoc co-design. In *First International Conference on Embedded Systems & Critical Applications (ICESCA'08), Tunis, Tunisia*, 2008.

[72] Alain Demeure and Yannick Del Gallo. An array approach for signal processing design. In *Sophia-Antipolis conference on Micro-Electronics (SAME'98), System-on-Chip Session, France*, October 1998.

[73] Jack B. Dennis. First version of a data flow procedure language. In *Programming Symposium, LNCS 19, Springer Verlag*, pages 362–376, 1974.

[74] Emil Dumitrescu, Alain Girault, Hervé Marchand, and Éric Rutten. Multicriteria optimal reconfiguration of fault-tolerant real-time tasks. In *Workshop on Discrete Event Systems, WODES'10*, pages 366–373, Berlin, Allemagne, August 2010. IFAC.

[75] Marc Duranton. The challenges for high performance embedded systems. In *DSD'06*, pages 3–7, 2006.

[76] Marc Duranton, Sami Yehia, Bjorn De Sutter, Koen De Bosschere, Albert Cohen, Babak Falsafi, Georgi Gaydadjiev, Manolis Katevenis, Jonas Maebe, Harm Munk, Nacho Navarro, Alex Ramirez, Olivier Temam, and Mateo Valero. The hipeac vision. Report, European Network of Excellence on High Performance and Embedded Architecture and Compilation, 2010. URL http://www.hipeac.net/system/files/hipeacvision.pdf.

[77] Marc Duranton, David Black-Schaffer, Sami Yehia, and Koen De Bosschere. Computing Systems: Research Challenges Ahead The HiPEAC Vision 2011/2012. Report, European Network of Excellence on High Performance and Embedded Architecture and Compilation, 2011. URL http://www.hipeac.net/system/files/hipeac-roadmap2011.pdf.

[78] Hritam Dutta. *Synthesis and Exploration of Loop Accelerators for Systems-on-a-Chip*. Ph.d. thesis, Der Technischen Fakultät der Universität Erlangen-Nürnberg zur Erlangung des Grades, Erlangen, Germany, 2011.

[79] Johan Eker and Jorn W. Janneck. Cal language report specification of the cal actor language. Technical Report UCB/ERL M03/48, EECS Department, University of California, Berkeley, 2003. URL http://www.eecs.berkeley.edu/Pubs/TechRpts/2003/4186.html.

[80] Johan Eker, Jorn Janneck, Edward A. Lee, Jie Liu, Xiaojun Liu, Jozsef Ludvig, Sonia Sachs, and Yuhong Xiong. Taming heterogeneity - the ptolemy approach. *Proceedings of the IEEE*, 91(1):127–144, January 2003.

[81] Hadi Esmaeilzadeh, Emily Blem, Renee St. Amant, Karthikeyan Sankaralingam, and Doug Burger. Dark silicon and the end of multicore scaling. *SIGARCH Comput. Archit. News*, 39:365–376, June 2011.

[82] Joachim Falk, Joachim Keinert, Christian Haubelt, Jürgen Teich, and Christian Zebelein. Integrated modeling using finite state machines and dataflow graphs. In Shuvra S. Bhattacharyya, Ed F. Deprettere, Rainer Leupers, and Jarmo Takala, editors, *Handbook of Signal Processing Systems*, pages 1041–1075. Springer US, 2010.

[83] Paul Feautrier. Array expansion. In *Proceedings of the Second International Conference on Supercomputing*, St. Malo, France, 1988.

[84] Paul Feautrier. Dataflow analysis of array and scalar references. *International Journal of Parallel Programming*, 20(1):23–53 (or 23–52??), 1991.

[85] Paul Feautrier. Some efficient solutions to the affine scheduling problem. part ii. multidimensional time. *International Journal of Parallel Programming*, 21(6): 389–420, 1992.

[86] Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2):374–382, April 1985. ISSN 0004-5411. doi: 10.1145/3149.214121. URL http://doi.acm.org/10.1145/3149.214121.

[87] Samuel H. Fuller and Lynette I. Millett. Computing performance: Game over or next level? *Computer*, 44(1):31–38, January 2011. ISSN 0018-9162.

[88] Edgar Gabriel, Graham E. Fagg, George Bosilca, Thara Angskun, Jack J. Dongarra, Jeffrey M. Squyres, Vishal Sahay, Prabhanjan Kambadur, Brian Barrett, Andrew Lumsdaine, Ralph H. Castain, David J. Daniel, Richard L. Graham, and Timothy S. Woodall. Open MPI: Goals, concept, and design of a next generation MPI implementation. In *Proceedings, 11th European PVM/MPI Users' Group Meeting*, pages 97–104, Budapest, Hungary, September 2004.

[89] A. Gamatié and T. Gautier. Modeling of modular avionics architectures using the synchronous language signal. In *Proceedings of the Work In Progress session, 14th Euromicro Conference on Real Time Systems, ECRTS'02*, pages 25–28, 2002.

[90] A. Gamatié and T. Gautier. The signal approach to the design of system architectures. In *Engineering of Computer-Based Systems, 2003. Proceedings. 10th IEEE International Conference and Workshop on the*, pages 80–88. IEEE, 2003.

[91] A. Gamatié, T. Gautier, and L. Besnard. Modeling of avionics applications and performance evaluation techniques using the synchronous language signal. *proceedings of Synchronous Languages, Applications, and Programming (SLAP'03). Portugal*, 2003.

[92] Abdoulaye Gamatié. *Designing Embedded Systems with the Signal Programming Language: Synchronous, Reactive Specification*. Springer, New York, 2009. ISBN 978-1-4419-0940-4.

[93] Abdoulaye Gamatié. Design of Streaming Applications on MPSoCs using Abstract Clocks. In *Design, Automation and Test in Europe Conference (DATE'2012)*, Dresden, Allemagne, 2012.

[94] Abdoulaye Gamatié and Thierry Gautier. Synchronous Modeling of Modular Avionics Architectures using the SIGNAL Language. Rapport de recherche RR-4678, INRIA, 2002. URL http://hal.inria.fr/inria-00071907/en/.

[95] Abdoulaye Gamatié and Thierry Gautier. The signal synchronous multiclock approach to the design of distributed embedded systems. *IEEE Trans. Parallel Distrib. Syst.*, 21(5):641–657, 2010.

[96] Abdoulaye Gamatié and Laure Gonnord. Static analysis of synchronous programs in signal for efficient design of multi-clocked embedded systems. In *LCTES*, pages 71–80, 2011.

[97] Abdoulaye Gamatié, Christian Brunette, Romain Delamare, Thierry Gautier, and Jean-Pierre Talpin. A modeling paradigm for integrated modular avionics design. In *Proceedings of the 32nd EUROMICRO Conference on Software Engineering and Advanced Applications*, pages 134–143, 2006.

[98] Abdoulaye Gamatié, Thierry Gautier, and Paul Le Guernic. Towards static analysis of Signal programs using interval techniques. In *Synchronous Languages, Applications, and Programming (SLAP'06)*, March 2006.

[99] Abdoulaye Gamatié, Thierry Gautier, Paul Le Guernic, and Jean-Pierre Talpin. Polychronous design of embedded real-time applications. *ACM Trans. Softw. Eng. Methodol.*, 16(2), 2007.

[100] Abdoulaye Gamatié, Thierry Gautier, and Loïc Besnard. An Interval-Based Solution for Static Analysis in the Signal Language. In *15th Annual IEEE International Conference and Workshop on Engineering of Computer Based Systems (ECBS'2008), Belfast, Northern Ireland*, pages 182–190, April 2008.

[101] Abdoulaye Gamatié, Éric Rutten, and Huafeng Yu. A model for the mixed-design of data-intensive and control-oriented embedded systems. Research report 6589, INRIA, France, July 2008. URL http://hal.inria.fr/inria-00293909/en.

[102] Abdoulaye Gamatié, Éric Rutten, Huafeng Yu, Pierre Boulet, and Jean-Luc Dekeyser. Synchronous modeling and analysis of data-intensive applications. *EURASIP Journal of Embedded Systems*, 2008, 2008.

[103] Abdoulaye Gamatié, Éric Rutten, Huafeng Yu, Pierre Boulet, and Jean-Luc Dekeyser. Model-driven engineering and formal validation of high-performance embedded systems. *Scalable Computing: Practice and Experience*, 10(2), 2009.

[104] Abdoulaye Gamatié, Huafeng Yu, Gwenaël Delaval, and Éric Rutten. A case study on controller synthesis for data-intensive embedded systems. In *6th IEEE International Conference on Embedded Software and Systems (ICESS'2009)*, pages 75–82, 2009.

[105] Abdoulaye Gamatié, Vlad Rusu, and Éric Rutten. Operational semantics of the marte repetitive structure modeling concepts for data-parallel applications design. In *ISPDC*, pages 25–32, 2010.

[106] Abdoulaye Gamatié, Sébastien Le Beux, Éric Piel, Rabie Ben Atitallah, Anne Etien, Philippe Marquet, and Jean-Luc Dekeyser. A model-driven design framework for massively parallel embedded systems. *ACM Trans. Embedded Comput. Syst.*, 10(4):39, 2011.

[107] Jean-Luc Gaudiot, Thomas DeBoni, John Feo, A. P. Wim Böhm, Walid A. Najjar, and Patrick Miller. The sisal project: Real world functional programming. In *Compiler Optimizations for Scalable Parallel Systems Languages*, pages 45–72, 2001.

[108] Thierry Gautier and Paul Le Guernic. Code generation in the Sacres project. In *Safety-critical Systems Symposium, SSS'99, Springer.* Huntingdon, UK, February 1999.

[109] Andreas Gerstlauer. Host-compiled simulation of multi-core platforms. In *Proceedings of the 21st IEEE International Symposium on Rapid System Prototyping, RSP 2010, Fairfax, VA, USA,*, pages 1–6, 2010.

[110] Amir Hossein Ghamarian, Marc Geilen, Sander Stuijk, Twan Basten, Bart D. Theelen, Mohammad Reza Mousavi, A. J. M. Moonen, and Marco Bekooij. Throughput analysis of synchronous data flow graphs. In *ACSD*, pages 25–36. IEEE Computer Society, 2006.

[111] Alain Girault. A survey of automatic distribution method for synchronous programs. In F. Maraninchi, M. Pouzet, and V. Roy, editors, *International Workshop on Synchronous Languages, Applications and Programs, SLAP'05*, ENTCS, Edinburgh, UK, April 2005. Elsevier Science.

[112] Alain Girault, Lee Bilung, and Edward .A. Lee. Hierarchical finite state machines with multiple concurrency models. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 18(6):742 –760, jun 1999.

[113] Alain Girault, Xavier Nicollin, and Marc Pouzet. Automatic rate desynchronization of embedded reactive programs. *ACM Transaction on Embedded Computing Systems*, 5(3):687–717, August 2006.

[114] Calin Glitia and Pierre Boulet. Interaction between inter-repetition dependences and high-level transformations in array-ol. In *Conference on Design and Architectures for Signal and Image Processing 2009*, Sophia Antipolis, France, September 2009.

[115] Calin Glitia, Philippe Dumont, and Pierre Boulet. Array-ol with delays, a domain specific specification language for multidimensional intensive signal processing. *Multidimensional Systems and Signal Processing*, 21:105–131, 2010.

[116] Calin Glitia, Pierre Boulet, Eric Lenormand, and Michel Barreteau. Repetitive model refactoring strategy for the design space exploration of intensive signal processing applications. *J. of Systems Architecture*, 57:815 – 829, Oct. 2011.

[117] Calin Glitia, Julien DeAntoni, Frédéric Mallet, Jean-Vivien Millo, Pierre Boulet, and Abdoulaye Gamatié. Progressive and Explicit Refinement of Scheduling for Multidimensional Data-Flow Applications using UML Marte. *Design Automation for Embedded Systems*, 16(2), June 2012.

[118] Thierry Grandpierre. Modélisation d'architectures parallèles hétérogènes pour la génération automatique d'exécutifs distribués temps réel optimisés. In *Thèse de l'Université de Paris-Sud, U.F.R. Scientifique d'Orsay*, November 2000.

[119] Thierry Grandpierre and Y. Sorel. From algorithm and architecture specifications to automatic generation of distributed real-time executives: a seamless flow of graphs transformations. In *MEMOCODE2003, Formal Methods and Models for Codesign Conference, Mont Saint-Michel, France*, Juin 2003.

[120] Radu Grosu, Ingolf Krüger, and Thomas Stauner. Hybrid sequence charts. In *3th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC 2000)*, pages 104 –111, 2000.

[121] Jing Guo, Antonio Wendell De Oliveira Rodrigues, Jerarajan Thiyagalingam, Frédéric Guyomarch, Pierre Boulet, and Sven-Bodo Scholz. Harnessing the Power of GPUs without Losing Abstractions in SaC and ArrayOL: A Comparative Study. In *HIPS 2011, 16th International Workshop on High-Level Parallel Programming Models and Supportive Environments*, Anchorage (Alaska) United States, 05 2011. URL http://hal.inria.fr/inria-00569100/en/.

[122] Rajiv Gupta, Santosh Pande, Kleanthis Psarris, and Vivek Sarkar. Compilation techniques for parallel systems. *Parallel Computing*, 25(13–14):1741–1783, 1999.

[123] Sumit Gupta, Nikil Dutt, Rajesh Gupta, and Alex Nicolau. Spark : A high-lev l synthesis framework for applying parallelizing compiler transformations. In *Proc. of the 16th International Conference on VLSI Design, 2003*.

[124] George Hagen and Cesare Tinelli. Scaling up the formal verification of lustre programs with smt-based techniques. In *FMCAD '08: Proceedings of the 2008 International Conference on Formal Methods in Computer-Aided Design*, pages 1–9, Piscataway, NJ, USA, 2008. IEEE Press. ISBN 978-1-4244-2735-2.

[125] Olivier Hainque. *Etude d'un environnement d'exécution temps-réel, distribué et tolérant aux pannes pour le modèle synchrone.* Ph.d. thesis, Juin 2000.

[126] Nicolas Halbwachs and Siwar Baghdadi. Synchronous modelling of asynchronous systems. In *Proceedings of the Second International Conference on Embedded Software*, EMSOFT '02, pages 240–251. Springer-Verlag, 2002.

[127] Nicolas Halbwachs and Louis Mandel. Simulation and verification of asynchronous systems by means of a synchronous model. In *Proceedings of the Sixth International Conference on Application of Concurrency to System Design*, pages 3–14. IEEE Computer Society, 2006.

[128] Nicolas Halbwachs and Mathias Péron. Discovering properties about arrays in simple programs. *SIGPLAN Not.*, 43:339–348, June 2008.

[129] Nicolas Halbwachs and Daniel Pilaud. Use of a real-time declarative language for systolic array design and simulation. In *International Workshop on Systolic Arrays*, Oxford, jul 1986.

[130] Nicolas Halbwachs, Paul Caspi, Pascal Raymond, and Daniel Pilaud. The synchronous dataflow programming language Lustre. In *IEEE, vol.79(9)*, pages 1305–1320, September 1991.

[131] Nicolas Halbwachs, Pascal Raymond, and Christophe Ratel. Generating efficient code from data-flow programs. In *Third International Symposium on Programming Language Implementation and Logic Programming*, Passau, Germany, August 1991.

[132] Nicolas Halbwachs, Fabienne Lagnier, and Christophe Ratel. Programming and verifying real-time systems by means of the synchronous data-flow programming language Lustre. *IEEE Transactions on Software Engineering, Special Issue on the Specification and Analysis of Real-Time Systems*, September 1992.

[133] Christian Haubelt, Joachim Falk, Joachim Keinert, Thomas Schlichter, Andreas Hadert Martin Streubühr, Andreas Deyhle, and Jürgen Teich. A systemc-based design methodology for digital signal processing systems. *EURASIP Journal on Embedded Systems*, 2007, 2007. Article ID 47580.

[134] Damien Hedde and Frédéric Pétrot. A non intrusive simulation-based trace system to analyse multiprocessor systems-on-chip software. In *International Symposium on Rapid System Prototyping*, pages 106–112. IEEE, 2011. ISBN 978-1-4577-0658-5.

[135] Damien Hedde, Pierre-Henri Horrein, Frédéric Petrot, Robin Rolland, and Franck Rousseau. A mpsoc prototyping platform for flexible radio applications. In *Euromicro DSD'09*, pages 559–566. IEEE Computer Society, 2009.

[136] High Performance Fortran Forum. High performance fortran language specification version 2.0. Technical report, Department of Computer Science, Rice University, 1997.

[137] Qubo Hu, Per Gunnar Kjeldsberg, Arnout Vandecappelle, Martin Palkovic, and Francky Catthoor. Incremental hierarchical memory size estimation for steering of loop transformations. *ACM Trans. on Design Automation of Electronic Sys.*, 12(50), Sept. 2007.

[138] The MathWorks Inc. Matlab r2011b documentation, 2011. URL `http://www.mathworks.fr/help/index.html`.

[139] International Technology Roadmap for Semiconductors, 2008. URL http://www.itrs.net. ITRS 2008 Update Overview.

[140] Kenneth E. Iverson. A personal view of apl. *IBM Systems Journal*, 30(4):582–593, 1991.

[141] Erwan Jahier, Nicolas Halbwachs, and Pascal Raymond. Synchronous modeling and validation of priority inheritance schedulers. In *Proceedings of the 12th International Conference on Fundamental Approaches to Software Engineering: Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2009*, FASE '09, pages 140–154. Springer-Verlag, 2009.

[142] Bertrand Jeannet. Dynamic partitioning in linear relation analysis. application to the verification of reactive systems. *Formal Methods in System Design*, 23(1): 5–37, July 2003.

[143] Kwangok Jeong, A.B. Kahng, B. Lin, and K. Samadi. Accurate machine-learning-based on-chip router modeling. *Embedded Systems Letters, IEEE*, 2 (3):62 –66, sept. 2010.

[144] Bijoy A. Jose and Sandeep K. Shukla. Mricdf: A polychronous model for embedded software synthesis. In Sandeep K. Shukla and Jean-Pierre Talpin, editors, *Synthesis of Embedded Software*, pages 173–199. Springer US, 2010.

[145] Bijoy A. Jose, Jason Pribble, and Sandeep K. Shukla. Faster software synthesis using actor elimination techniques for polychronous formalism. In *ACSD*, pages 147–156, 2010.

[146] Bijoy A. Jose, Abdoulaye Gamatié, Julien Ouy, and Sandeep K. Shukla. SMT Based False Causal loop Detection during Code Synthesis from Polychronous Specifications. In *ACM/IEEE Ninth International Conference on Formal Methods and Models for Codesign (MEMOCODE)*, pages 109 –118, 2011.

[147] Bijoy Anthony Jose, Abdoulaye Gamatié, Matthew Kracht, and Sandeep Kumar Shukla. Improved False Causal Loop Detection in Polychronous Specificationof Embedded Software. Research report, 2011. URL http://hal.inria.fr/inria-00637582.

[148] J. A. Kahle, M. N. Day, H. P. Hofstee, C. R. Johns, T. R. Maeurer, and D. Shippy. Introduction to the cell multiprocessor. *IBM J. Res. Dev.*, 49:589–604, July 2005.

[149] Gilles Kahn. The semantics of simple language for parallel programming. In *IFIP Congress*, pages 471–475, 1974.

[150] Joachim Keinert, Joachim Falk, Christian Haubelt, and Jürgen Teich. Actor-oriented modeling and simulation of sliding window image processing algorithms. In *Embedded Systems for Real-Time Multimedia, 2007. ESTIMedia 2007. IEEE/ACM/IFIP Workshop on*, pages 113 –118, oct. 2007.

[151] Joachim Keinert, Martin Streubuhr, Thomas Schlichter, Joachim Falk, Jens Gladigau, Christian Haubelt, J&uhorbar;rgen Teich, and Michael Meredith. Systemcodesigner – an automatic esl synthesis approach by design space exploration and behavioral synthesis for streaming applications. *ACM Trans. Des. Autom. Electron. Syst.*, 14:1:1–1:23, January 2009.

[152] Peter Kogge, Keren Bergman, Shekhar Borkar, Dan Campbell, William Carlson, William Dally, Monty Denneau, Paul Franzon, William Harrod, Kerry Hill, Jon Hiller, Sherman Karp, Stephen Keckler, Dean Klein, Robert Lucas, Mark Richards, Al Scarpelli, Steven Scott, Allan Snavely, Thomas Sterling, R. Stanley Williams, and Katherine Yelick. ExaScale Computing Study: Technology Challenges in Achieving Exascale Systems. Technical report, 2008. URL http://www.cse.nd.edu/Reports/2008/TR-2008-13.pdf. Report, Peter Kogge, Editor & Study Lead.

[153] David Kolson, Alexandru Nicolau, and Nikil Dutt. Elimination of redundant memory traffic in high-level synthesis. *IEEE Trans. on Comp-aided Design*, 15: 1354–1363, 1996.

[154] Apostolos Kountouris and Paul Le Guernic. Profiling of Signal programs and its application in the timing evaluation of design implementations. In *IEE Colloq. on HW-SW Cosynthesis for Reconfigurable Systems, IEE*, pages 6/1–6/9. HP Labs, Bristol, UK, February 1996.

[155] Rakesh Kumar, Timothy G. Mattson, Gilles Pokam, and Rob Wijngaart. The case for message passing on many-core chips. In Michael Hübner and Jürgen Becker, editors, *Multiprocessor System-on-Chip*, pages 115–123. Springer New York, 2011.

[156] Ouassila Labbani. *Modélisation à haut niveau du contrôle dans des applications de traitement systématique á parallélisme massif.* Ph.d. thesis, Université de Lille 1, France, November 2006. URL http://www.lifl.fr/west/publi/Labb06phd.pdf.

[157] Chris Lattner and Vikram Adve. Llvm: A compilation framework for lifelong program analysis & transformation. In *Proceedings of the International Symposium on Code Generation and Optimization (CGO'04): feedback-directed and runtime optimization*, pages 75–. IEEE Computer Society, 2004.

[158] Bernard Le Goff. *Inférence de contrôle hiérarchique: application au temps réel*. Ph.d. thesis, Université de Rennes I, IFSIC, France, 1989.

[159] Paul Le Guernic. Signal : Description algébrique des flots de signaux. In *Architecture des machines et systèmes informatiques. Actes du congrès de l'Afcet*, pages 243–252, November 1982. Hommes et Techniques.

[160] Paul Le Guernic, Albert Benveniste, and Thierry Gautier. SIGNAL:Un langage pour le traitement du signal. Research Report RR-0206, INRIA/IRISA, 1983. URL http://hal.inria.fr/inria-00076352.

[161] Paul Le Guernic, Albert Benveniste, Patricia Bournai, and Thierry Gautier. SIGNAL: A Data Flow-Oriented Language for Signal Processing. *IEEE Transactions on Acoustics, Speech and Signal Procesing*, ASSP-34(2), April 1986.

[162] Paul Le Guernic, Thierry Gautier, Michel Le Borgne, and Claude Le Maire. Programming real-time applications with Signal. *Proceedings of the IEEE*, 79(9): 1321–1336, 1991.

[163] Paul Le Guernic, Jean-Pierre Talpin, and Jean-Christophe Le Lann. Polychrony for System Design. *Journal for Circuits, Systems and Computers*, 12(3):261–304, April 2003.

[164] Le Verge, Hervé and Mauras, Christophe and Quinton, Patrice. The alpha language and its use for the design of systolic arrays. *The Journal of VLSI Signal Processing*, 3:173–182, 1991.

[165] Edward A. Lee. Computing needs time. *Commun. ACM*, 52(5):70–79, May 2009. ISSN 0001-0782. doi: 10.1145/1506409.1506426. URL http://doi.acm.org/10.1145/1506409.1506426.

[166] Edward A. Lee and David G. Messerschmitt. Synchronous data flow: Describing signal processing algorithm for parallel computation. In *COMPCON*, pages 310–315, 1987.

[167] Edward A. Lee and Alberto Sangiovanni-vincentelli. A framework for comparing models of computation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 17:1217–1229, 1998.

[168] Tsing-Fa Lee, Allen C.-H. Wu, Youn-Long Lin, and Daniel D. Gajski. A transformation-based method for loop folding. *IEEE Trans. on CAD of Integrated Circuits and Systems*, pages 439–450, 1994.

[169] Christian Lengauer. Loop parallelization in the polytope model. In *Proceedings of the 4th International Conference on Concurrency Theory*, CONCUR '93, pages 398–416, London, UK, 1993. Springer-Verlag. ISBN 3-540-57208-2.

[170] Xin Li and Reinhard von Hanxleden. Multithreaded reactive programming - the kiel esterel processor. *IEEE Trans. Computers*, 61(3):337–349, 2012.

[171] C. L. Liu and James W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM*, 20(1):46–61, January 1973. ISSN 0004-5411.

[172] Nestor Lopez, Marianne Simonot, and Véronique Donzeau-Gouge. A methodological process for the design of a large system: two industrial case-studies. *ENTCS*, 66(2), 2002.

[173] Martin Lukasiewycz, Michael Glaß, Felix Reimann, and Jürgen Teich. Opt4J - A Modular Framework for Meta-heuristic Optimization. In *Proc. of the Genetic and Evolutionary Computing Conference (GECCO 2011)*, .

[174] Martin Lukasiewycz, Martin Streubühr, Michael Glaß, Christian Haubelt, and Jürgen Teich. Combined system synthesis and communication architecture exploration for mpsocs. In *Proc. of the Conference on Design, Automation and Test in Europe (DATE'09)*, .

[175] Yue Ma, J.-P. Talpin, and T. Gautier. Virtual prototyping aadl architectures in a polychronous model of computation. In *Formal Methods and Models for Co-Design, 2008. MEMOCODE 2008. 6th ACM/IEEE International Conference on*, pages 139 –148, June 2008.

[176] Olivier Maffeïs. *Ordonnancements de graphes de flots synchrones : application à la mise en œuvre de Signal.* Ph.d. thesis, Université de Rennes I, IFSIC, France, January 1993.

[177] Grigorios Magklis, Greg Semeraro, David H. Albonesi, Steven G. Dropsho, Sandhya Dwarkadas, and Michael L. Scott. Dynamic frequency and voltage scaling for a multiple-clock-domain microprocessor. *Micro, IEEE*, 23(6):62 – 68, November 2003.

[178] Frédéric Mallet. Clock constraint specification language: specifying clock constraints with uml/marte. *Innovations in Systems and Software Engineering*, 4: 309–314, 2008.

[179] Florence Maraninchi and Yann Rémond. Mode-automata: a new domain-specific construct for the development of safe critical systems. *Science of Computer Programming*, 46(1-2):219–254, 2003.

[180] Hervé Marchand, Patricia Bournai, Michel Le Borgne, and Paul Le Guernic. Synthesis of Discrete-Event Controllers based on the Signal Environment. *Discrete Event Dynamic System: Theory and Applications*, 10(4):325–346, October 2000.

[181] César Marcon, Ney Calazans, Edson Moreno, Fernando Moraes, Fabiano Hessel, and Altamiro Susin. Cafes: A framework for intrachip application modeling and communication architecture design. *Journal of Parallel and Distributed Computing*, 71(5):714 – 728, 2011.

[182] Christophe Mauras. *Alpha : un langage équationnel pour la conception et la programmation d'architectures parallèles synchrones*. Ph.d. thesis, Université de Rennes I, France, December 1989.

[183] Message Passing Interface Forum. MPI Documents. http://www.mpi-forum.org/docs/docs.html, 2009.

[184] Olivier Michel. Design and implementation of 8 1/2 , a declarative data-parallel language. Technical report, Computer Languages, 1996.

[185] Lionel Morel. Array iterators in lustre: From a language extension to its exploitation in validation. *EURASIP Journal on Embedded Systems*, 2007.

[186] Mohammad R. Mousavi, Paul Le Guernic, Jean-Pierre Talpin, Sandeep K. Shukla, and Twan Basten. Modeling and validating globally asynchronous design in synchronous frameworks. In *DATE*, pages 384–389, 2004. ISBN 0-7695-2085-5-1.

[187] Jens Muttersbach, Thomas Villiger, Hubert Kaeslin, Norbert Felber, and Wolfgang Fichtner. Globally-Asynchronous Locally-Synchronous Architectures to Simplify the Design of On-chip Systems. In *12th IEEE International ASIC/SOC Conference*, Washington DC, USA, 1999.

[188] Mirabelle Nebut. Specification and analysis of synchronous reactions. *Formal Aspects of Computing*, 16(3):263–291, august 2004.

[189] Stephen Neuendorffer and Edward Lee. Hierarchical reconfiguration of dataflow models. In *2nd Int'l Conf. on Formal Methods and Models for Co-Design (MEMOCODE'04)*, pages 179–188, june 2004.

[190] Bradford Nichols, Dick Buttlar, and Jacqueline Proulx Farrell. *Pthreads programming*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 1996. ISBN 1-56592-115-1.

[191] John Nickolls, Ian Buck, Michael Garland, and Kevin Skadron. Scalable parallel programming with cuda. *Queue*, 6:40–53, March 2008.

[192] Bosko Nikolic, Zaharije Radivojevic, Jovan Djordjevic, and Veljko Milutinovic. A survey and evaluation of simulators suitable for teaching courses in computer architecture and organization. *Education, IEEE Transactions on*, 52(4):449 –458, nov. 2009.

[193] Robert W. Numrich and John Reid. Co-array fortran for parallel programming. *SIGPLAN Fortran Forum*, 17:1–31, August 1998.

[194] Object Management Group. A UML profile for MARTE, 2012. http://www.omgmarte.org.

[195] OpenMP Architecture Review Board. The OpenMP API specification for parallel programming. http://openmp.org, 2009.

[196] Claire Pagetti, Julien Forget, Frédéric Boniol, Mikel Cordovilla, and David Lesens. Multi-task implementation of multi-periodic synchronous programs. *Discrete Event Dynamic Systems*, 21(3):307–338, September 2011. ISSN 0924-6703.

[197] P. R. Panda, F. Catthoor, N. D. Dutt, K. Danckaert, E. Brockmeyer, C. Kulkarni, A. Vandercappelle, and P. G. Kjeldsberg. Data and memory optimization techniques for embedded systems. *ACM Trans. on Design Automation of Electronic Sys.*, 6:149–206, April 2001.

[198] Aashish Pant, Puneet Gupta, and Mihaela van der Schaar. Software adaptation in quality sensitive applications to deal with hardware variability. In *Proceedings of the 20th symposium on Great lakes symposium on VLSI*, GLSVLSI '10, pages 85–90. ACM, 2010.

[199] Aashish Pant, Puneet Gupta, and Mihaela van der Schaar. Appadapt: Opportunistic application adaptation in presence of hardware variation. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, PP(99):1 –11, 2011.

[200] Joonseok Park, Pedro C. Diniz, and K. R. Shesha Shayee. Performance and area modeling of complete FPGA designs in the presence of loop transformations. *IEEE Trans. Comput.*, 53:1420–1435, November 2004.

[201] Valentin Perrelle and Nicolas Halbwachs. An analysis of permutations in arrays. In *VMCAI*, pages 279–294, 2010.

[202] Carl Adam Petri. *Kommunikation mit Automaten*. Ph.d. thesis, Darmstadt University of Technology, Germany, 1962.

[203] Frederic Petrot, Nicolas Fournel, Patrice Gerin, Marius Gligor, Mian-Muhammed Hamayun, and Hao Shen. On mpsoc software execution at the transaction level. *IEEE Des. Test*, 28:32–43, May 2011.

[204] John Plaice. Multidimensional lucid: Design, semantics and implementation. In *In Distributed Communities on the Web: Third International Workshop, DCW 2000*. Springer, 2000.

[205] Amir Pnueli. Specification and development of reactive systems (invited paper). In *IFIP Congress*, pages 845–858, 1986.

[206] Dumitru Potop-Butucaru, Stephen A. Edwards, and Gerard Berry. *Compiling Esterel*. Springer Publishing Company, Incorporated, 1st edition, 2007. ISBN 0387706267, 9780387706269.

[207] Dumitru Potop-Butucaru, Akramul Azim, and Sebastian Fischmeister. Semantics-preserving implementation of synchronous specifications over dynamic tdma distributed architectures. In *EMSOFT*, pages 199–208, 2010.

[208] Yue Qian, Zhonghai Lu, and Wenhua Dou. Analysis of communication delay bounds for network on chips. In *Proceedings of the 2009 Asia and South Pacific Design Automation Conference (ASP-DAC'09)*, pages 7–12. IEEE Press, 2009. ISBN 978-1-4244-2748-2.

[209] Imran Rafiq Quadri, Huafeng Yu, Abdoulaye Gamatié, Éric Rutten, Samy Meftali, and Jean-Luc Dekeyser. Targeting reconfigurable fpga based socs using the uml marte profile: from high abstraction levels to code generation. *IJES*, 4 (3/4):204–224, 2010.

[210] I.R. Quadri, A. Gamatié, P. Boulet, and J.L. Dekeyser. Modeling of configurations for embedded system implementations in marte. In *1st workshop on Model Based Engineering for Embedded Systems Design-Design, Automation and Test in Europe (DATE'2010)*, 2010.

[211] Talal Rahwan, Sarvapali Ramchurn, Nicholas Jennings, and Andrea Giovannucci. An anytime algorithm for optimal coalition structure generation. *J. of Artificial Intelligence Research (JAIR)*, 34:521–567, April 2009.

[212] Keith H. Randall. *Cilk: Efficient Multithreaded Computing*. Ph.d. thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, May 1998.

[213] Parthasarathy Ranganathan. From microprocessors to nanostores: Rethinking data-centric systems. *Computer*, 44(1):39 –48, jan. 2011.

[214] Frédéric Rocheteau and Nicolas Halbwachs. Implementing reactive programs on circuits: A hardware implementation of lustre. In *Proceedings of the Real-Time: Theory in Practice, REX Workshop*, pages 195–208. Springer-Verlag, 1992.

[215] Éric Rutten and Florent Martinez. Signalgti: Implementing task preemption and time intervals in the synchronous data flow language signal. In *In proceedings of the 7th Euromicro Workshop on Real Time Systems*, pages 176–183. IEEE Publ, 1995.

[216] Alberto Sangiovanni-Vincentelli, Luca Carloni, Fernando De Bernardinis, and Marco Sgroi. Benefits and challenges for platform-based design. In *Proceedings of the 41st annual Design Automation Conference*, DAC'04, pages 409–414, 2004. ISBN 1-58113-828-8.

[217] Alberto L. Sangiovanni-Vincentelli, Marco Sgroi, and Luciano Lavagno. Formal models for communication-based design. In *Proceedings of the 11th International Conference on Concurrency Theory (CONCUR'00)*, pages 29–47. Springer-Verlag, 2000.

[218] Sven-Bodo Scholz. Single assignment c: efficient support for high-level array operations in a functional setting. *J. Funct. Program.*, 13(6):1005–1059, 2003.

[219] Irina M. Smarandache, Thierry Gautier, and Paul Le Guernic. Validation of Mixed SIGNAL-ALPHA Real-Time Systems through Affine Calculus on Clock Synchronisation Constraints. In *Proceedings of the World Congress on Formal Methods (FM'99)*, pages 1364–1383. Springer-Verlag, 1999.

[220] Antoine Spicher, Olivier Michel, and Jean-Louis Giavitto. Spatial computing as intensional data parallelism. In *Proceedings of the 2010 Fourth IEEE International Conference on Self-Adaptive and Self-Organizing Systems Workshop*, SASOW '10, pages 196–205, Washington, DC, USA, 2010. IEEE Computer Society. ISBN 978-0-7695-4229-4.

[221] Sathya Sriram and Shuvra S. Bhattacharyya. *Embedded multiprocessors: Scheduling and synchronization*. CRC Press, 2000.

[222] Stanford Streaming Supercomputer Project. Merrimac - Brook page, 2009. http://merrimac.stanford.edu/brook.

[223] Karsten Strehl and Lothar Thiele. Symbolic model checking of process networks using interval diagram techniques. In *ICCAD*, pages 686–692, 1998.

[224] Jean-Pierre Talpin, Abdoulaye Gamatié, David Berner Le Dez, and Paul Le Guernic. Hard real-time implementation of embedded software in java. In *FIDJI'2003. Lectures Notes in Computer Science*. Springer, 2003.

[225] Jean-Pierre Talpin, Christian Brunette, Thierry Gautier, and Abdoulaye Gamatié. Polychronous mode automata. In *Proceedings of the 6th ACM & IEEE International conference on Embedded software*, EMSOFT '06, pages 83–92. ACM, 2006.

[226] Andrew S. Tanenbaum and Maarten van Steen. *Distributed Systems: Principles and Paradigms (2nd ed.)*. Prentice Hall, 2007.

[227] B.D. Theelen, M.C.W. Geilen, T. Basten, J.P.M. Voeten, S.V. Gheorghita, and S. Stuijk. A scenario-aware data flow model for combined long-run average and worst-case performance analysis. In *4th Int'l Conf. on Formal Methods and Models for Co-Design (MEMOCODE'06)*, pages 185–194, july 2006.

[228] Scott Thibault. FPGA for DSP: A JPEG encoder case study, Sept. 2011. URL http://www.gmvhdl.com/fpga_for_dsp.html.

[229] William Thies, Michal Karczmarek, and Saman P. Amarasinghe. Streamit: A language for streaming applications. In *Proceedings of the 11th International Conference on Compiler Construction*, CC '02, pages 179–196. Springer-Verlag, 2002.

[230] Mark Thompson, Hristo Nikolov, Todor Stefanov, Andy D. Pimentel, Cagkan Erbas, Simon Polstra, and Ed F. Deprettere. A framework for rapid system-level exploration, synthesis, and programming of multimedia MP-SoCs. In *CODES+ISSS'07*, pages 9–14. ACM, 2007.

[231] Takao Toi, Toru Awashima, Masato Motomura, and Hideharu Amano. Time and space-multiplexed compilation challenges for dynamically reconfigurable processors. In *Circuits and Systems (MWSCAS), 2011 IEEE 54th International Midwest Symposium on*, pages 1 –4, aug. 2011.

[232] Nick Tredennick and Brion Shimamoto. The inevitability of reconfigurable systems. *Queue*, 1:34–43, October 2003.

[233] Lewis W. Tucker and George G. Robertson. Architecture and applications of the connection machine. *Computer*, 21:26–38, August 1988.

[234] Vincent Van Dongen, Guang Gao, and Qi Ning. A polynomial time method for optimal software pipelining. In Luc Bougé, Michel Cosnard, Yves Robert, and Denis Trystram, editors, *Parallel Processing: CONPAR 92-VAPP V*, volume 634 of *LNCS*, pages 613–624. Springer Berlin / Heidelberg, 1992.

[235] Variability Expedition. Variability-Aware Software for Efficient Computing with Nanoscale Devices, 2012. URL `http://variability.org`.

[236] Paulo Veríssimo. On the role of time in distributed systems. In *6th IEEE Workshop on Future Trends of Distributed Computer Systems (FTDCS'97), 29-31 October 1997, Tunis, Tunisia*, pages 316–323. IEEE Computer Society, 1997.

[237] Bruno Virlet, Xing Zhou, Jean Pierre Giacalone, Bob Kuhn, Maria J. Garzaran, and David Padua. Scheduling of stream-based real-time applications for heterogeneous systems. In *Proceedings of the 2011 SIGPLAN/SIGBED conference on Languages, compilers and tools for embedded systems*, LCTES '11, pages 1–10. ACM, 2011. ISBN 978-1-4503-0555-6.

[238] William W. Wadge and Edward A. Ashcroft. *LUCID, the dataflow programming language*. Academic Press Professional, Inc., San Diego, CA, USA, 1985. ISBN 0-12-729650-6.

[239] Robert A. Walker and Samit Chaudhuri. Introduction to the scheduling problem. *Design Test of Computers, IEEE*, 12(2):60 –69, summer 1995.

[240] Doran Wilde. The Alpha language. Technical Report 827, IRISA - INRIA, Rennes, 1994. available at `http://hal.inria.fr/inria-00074378`.

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

[242] Yang Yang, Marc Geilen, Twan Basten, Sander Stuijk, and Henk Corporaal. Automated bottleneck-driven design-space exploration of media processing systems. In *DATE'2010*, pages 1041–1046, 2010.

[243] Kathy Yelick, Luigi Semenzato, Geoff Pike, Carleton Miyamoto, Ben Liblit, Arvind Krishnamurthy, Paul Hilfinger, Susan Graham, David Gay, Phil Colella, and Alex Aiken. Titanium: A high-performance Java dialect. In *ACM 1998 Workshop on Java for High-Performance Network Computing*, New York, NY 10036, USA, 1998. ACM Press.

[244] Huafeng Yu. *A MARTE-Based Reactive Model for Data-Parallel Intensive Processing: Transformation Toward the Synchronous Model*. Ph.d. thesis, Université de Lille 1, France, 2008. URL `http://hal.inria.fr/tel-00497248`.

[245] Huafeng Yu, Abdoulaye Gamatié, Éric Rutten, and Jean-Luc Dekeyser. Safe Design of High-Performance Embedded Systems in a MDE framework. *Innovations in Systems and Software Engineering (ISSE)*, 4(3), 2008. NASA/Springer journal ISSE.

[246] Jun Zhu, Ingo Sander, and Axel Jantsch. Energy efficient streaming applications with guaranteed throughput on mpsocs. In *EMSOFT'08, Atlanta, GA, USA*, pages 119–128, 2008.

[247] Jun Zhu, Ingo Sander, and Axel Jantsch. Pareto efficient design for reconfigurable streaming applications on cpu/fpgas. In *DATE'2010*, pages 1035–1040, 2010.

[248] Eckart Zitzler, Lothar Thiele, Marco Laumanns, Carlos M. Fonseca, and Viviane Grunert da Fonseca. Performance assessment of multiobjective optimizers: An analysis and review. *IEEE Trans. on Evolutionary Computation*, 7:117–132, April 2003.

This document was typeset using the typographical look-and-feel `classicthesis` developed by André Miede. The style was inspired by Robert Bringhurst's seminal book on typography "*The Elements of Typographic Style*". `classicthesis` is available for both LaTeX and LyX:

`http://code.google.com/p/classicthesis/`