IN PARTNERSHIP WITH:
CNRS

Ecole normale supérieure de Lyon

Université Claude Bernard (Lyon 1)

Activity Report 2019

Project-Team CASH

Compilation and Analyses for Software and Hardware

IN COLLABORATION WITH: Laboratoire de l'Informatique du Parallélisme (LIP)

RESEARCH CENTER
Grenoble - Rhône-Alpes

THEME
Architecture, Languages and Compilation

Table of contents

Project-Team CASH

Creation of the Team: 2018 April 01, updated into Project-Team: 2019 June 01

Keywords:

Computer Science and Digital Science:

A1.1.1. - Multicore, Manycore
A1.1.2. - Hardware accelerators (GPGPU, FPGA, etc.)
A1.1.4. - High performance computing
A1.1.10. - Reconfigurable architectures
A1.1.12. - Non-conventional architectures
A2.1.1. - Semantics of programming languages
A2.1.6. - Concurrent programming
A2.1.7. - Distributed programming
A2.2.1. - Static analysis
A2.2.3. - Memory management
A2.2.4. - Parallel architectures
A2.2.6. - GPGPU, FPGA...
A2.2.8. - Code generation
A2.3.1. - Embedded systems
A2.5.3. - Empirical Software Engineering
A2.5.4. - Software Maintenance & Evolution

Other Research Topics and Application Domains:

B9.5.1. - Computer science

1. Team, Visitors, External Collaborators

Research Scientists

Christophe Alias [Inria, Researcher, HDR]
Ludovic Henrio [CNRS, Researcher, HDR]

Faculty Members

Matthieu Moy [Team leader, Univ Claude Bernard, Associate Professor, HDR]
Laure Gonnord [Univ Claude Bernard, Associate Professor, HDR]

PhD Students

Julien Braine [École Normale Supérieure de Lyon, PhD Student]
Paul Iannetta [École Normale Supérieure de Lyon, PhD Student]
Amaury Maille [École Normale Supérieure de Lyon, PhD Student, from Oct 2019]

Interns and Apprentices

Maximilien Dupont de Dinechin [Inria, from Jun 2019 until Jul 2019]
Amaury Maille [École Normale Supérieure de Lyon, until Sep 2019]
Julien Philippon [École Normale Supérieure de Lyon, from Jul 2019]
Julien Rudeau [Inria, from Apr 2019 until Aug 2019]

Administrative Assistants

Kadiatou Bangoura [Inria, Administrative Assistant, until Aug 2019]
Sophie Gerard [Inria, Administrative Assistant, from Sep 2019]

2. Overall Objectives

2.1. Introduction

Until 2006, the typical power-consumption of a chip remained constant for a given silicon area as the transistor size decreased (this evolution is referred to as Dennard scaling). In other words, energy efficiency was following an exponential law similar to Moore's law. This is no longer true, hence radical changes are needed to further improve power efficiency, which is the limiting factor for large-scale computing. Improving the performance under a limited energy budget must be done by rethinking computing systems at all levels: hardware, software, compilers, and runtimes.

On the hardware side, new architectures such as multi-core processors, Graphics Processing Units (GPUs), many-core and FPGA accelerators are introduced, resulting into complex heterogeneous platforms. In particular, FPGAs are now a credible solution for energy-efficient HPC. An FPGA chip can deliver the same computing power as a GPU for an energy budget 10 times smaller.

A consequence of this diversity and heterogeneity is that a given computation can be implemented in many different ways, with different performance characteristics. An obvious example is changing the degree of parallelism: this allows trading execution time for number of cores used. However, many choices are less obvious: for example, augmenting the degree of parallelism of a memory-bounded application will not improve performance. Most architectures involve a complex memory hierarchy, hence memory access patterns have a considerable impact on performance too. The design-space to be explored to find the best performance is much wider than it used to be with older architectures, and new tools are needed to help the programmer explore it. The problem is even stronger for FPGA accelerators, where programmers are expected to design a circuit for their application! Traditional synthesis tools take as input low-level languages like VHDL and Verilog. As opposed to this, high-level languages and hardware compilers (HLS, High-Level Synthesis, that takes as input a C or C-like language and produces a circuit description) are required.

One of the bottlenecks of performance and energy efficiency is data movement. The operational intensity (ratio computation/communication) must be optimized to avoid memory-bounded performance. Compiler analyses are strongly required to explore the trade-offs (operational intensity vs. local memory size, operational intensity vs. peak performance for reconfigurable circuits).

These issues are considered as one of the main challenges in the Hipeac roadmap [33] which, among others, cites the two major issues:

- Applications are moving towards global-scale services, accessible across the world and on all devices. Low power processors, systems, and communications are key to computing at this scale. (*Strategic Area 2, Data Center Computing*).
- Today data movement uses more power than computation. [...] To adapt to this change, we need to expose data movement in applications and optimize them at runtime and compile time and to investigate communication-optimized algorithms (*cross-cutting challenge 1, energy efficiency*).

2.2. Overall Objectives

The overall objective of the CASH team is to take advantage of the characteristics of the specific hardware (generic hardware, hardware accelerators, or reconfigurable chips) to *compile energy efficient software and hardware*. More precisely, the CASH team works on:

1. Definition of dataflow representations of parallel programs that can capture the parallelism at all levels: fine-grain vs. coarse-grain, data & task parallelism, programming language, and intermediate representation (Section 3.1).
2. Scalable and expressive static program analyses. CASH works on improving the scalability of analyses to allow a global analysis of large-scale programs, and on the expressiveness of analysis to find better program invariants. Analysis is performed both on the representation defined above and on general programs (Section 3.2).

3. Transformations from and to the dataflow representation, combining traditional tools dedicated to dataflow and specific methods like the polyhedral model (Section 3.3).
4. A high-level synthesis (HLS) tool, built on the above item (instantiated with the particularities of FPGAs) and a code generation tool (Section 3.4). This HLS tool focuses on early stages of compilation and rely on an external tool for the back-end.
5. A parallel and scalable simulation of hardware systems, which, combined with the preceding activity, will result in an end-to-end workflow for circuit design (Section 3.5).

To ensure the coherency and the correctness of our approach these different tasks will rely on a *precise definition of the manipulated languages and their semantics*. The formalization of the different representations of the programs and of the analyses will allow us to show that these different tasks will be performed with the same understanding of the program semantics.

Note that these directions are strongly tied together. We use 5 research directions for the sake of the presentation, but their complementarity enables each member of the team to share common research goals while having their own research directions. Most of our results contribute to several directions.

3. Research Program

3.1. Definition of dataflow representations of parallel programs

In the last decades, several frameworks have emerged to design efficient compiler algorithms. The efficiency of all the optimizations performed in compilers strongly relies on effective *static analyses* and *intermediate representations*. Dataflow models are a natural intermediate representation for hardware compilers (HLS) and more generally for parallelizing compilers. Indeed, dataflow models capture task-level parallelism and can be mapped naturally to parallel architectures. In a way, a dataflow model is a partition of the computation into processes and a partition of the flow dependences into channels. This partitioning prepares resource allocation (which processor/hardware to use) and medium-grain communications.

The main goal of the CASH team is to provide efficient analyses and the optimizing compilation frameworks for dataflow programming models. The results of the team relies on programming languages and representation of programs in which parallelism and dataflow play a crucial role. This first research direction aims at defining these dataflow languages and intermediate representations, both from a practical perspective (syntax or structure), and from a theoretical point of view (semantics). This first research direction thus defines the models on which the other directions will rely. It is important to note that we do not restrict ourselves to a strict definition of dataflow languages: more generally, we are interested in the parallel languages in which dataflow synchronization plays a significant role.

Intermediate dataflow model. The intermediate dataflow model is a representation of the program that is adapted for optimization and scheduling. It will be obtained from the analysis of a (parallel or sequential) program and should at some point be used for compilation. The dataflow model must specify precisely its semantics and parallelism granularity. It must also be analyzable with polyhedral techniques, where powerful concepts exist to design compiler analysis, e.g., scheduling or resource allocation. Polyhedral Process Networks [58] extended with a module system could be a good starting point. But then, how to fit non-polyhedral parts of the program? A solution is to hide non-polyhedral parts into processes with a proper polyhedral abstraction. This organization between polyhedral and non-polyhedral processes will be a key aspect of our medium-grain dataflow model. The design of our intermediate dataflow model and the precise definition of its semantics will constitute a reliable basis to formally define and ensure the correctness of algorithms proposed by CASH: compilation, optimizations and analyses.

Dataflow programming languages. Dataflow paradigm has also been explored quite intensively in programming languages. Indeed, there exists a large panel of dataflow languages, whose characteristics differ notably, the major point of variability being the scheduling of agents and their communications. There is indeed a continuum from the synchronous dataflow languages like Lustre [42] or Streamit [55], where the scheduling is fully static, and general communicating networks like KPNs [45] or RVC-Cal [25] where a dedicated runtime is responsible for scheduling tasks dynamically, when they *can* be executed. These languages share some similarities with actor languages that go even further in the decoupling of processes by considering them as independent reactive entities. Another objective of the CASH team is to study dataflow programming *languages*, their semantics, their expressiveness, and their compilation. The specificity of the CASH team is that these languages will be designed taking into consideration the compilation using polyhedral techniques. In particular, we will explore which dataflow constructs are better adapted for our static analysis, compilation, and scheduling techniques. In practice we want to propose high-level primitives to express data dependency, this way the programmer can express parallelism in a dataflow way instead of the classical communication-oriented dependencies. The higher-level more declarative point of view makes programming easier but also give more optimization opportunities. These primitives will be inspired by the existing works in the polyhedral model framework, as well as dataflow languages, but also in the actors and active object languages [32] that nowadays introduce more and more dataflow primitives to enable data-driven interactions between agents, particularly with *futures* [30], [37].

3.1.1. Expected Impact

Consequently, the impact of this research direction is both the usability of our representation for static analyses and optimizations performed in Sections 3.2 and 3.3, and the usability of its semantics to prove the correctness of these analyses.

3.1.2. Scientific Program

3.1.2.1. Short-term and ongoing activities.

We obtained preliminary experimental [22], [23], [38] and theoretical [43] results, exploring several aspects of dataflow models. The next step is to define accurately the intermediate dataflow model and to study existing programming and execution models:

- Define our medium-grain dataflow model. So far, a modular Polyhedral Process Networks appears as a natural candidate but it may need to be extended to be adapted to a wider range of applications. Precise semantics will have to be defined for this model to ensure the articulation with the activities discussed in Section 3.3.
- Study precisely existing dataflow languages, their semantics, their programmability, and their limitations.

3.1.2.2. Medium-term activities.

In a second step, we will extend the existing results to widen the expressiveness of our intermediate representation and design new parallelism constructs. We will also work on the semantics of dataflow languages:

- Propose new stream programming models and a clean semantics where all kinds of parallelisms are expressed explicitly, and where all activities from code design to compilation and scheduling can be clearly expressed.
- Identify a core language that is rich enough to be representative of the dataflow languages we are interested in, but abstract and small enough to enable formal reasoning and proofs of correctness for our analyses and optimizations.

3.1.2.3. Long-term activities.

In a longer-term vision, the work on semantics, while remaining driven by the applications, would lead to to more mature results, for instance:

- Design more expressive dataflow languages and intermediate representations which would at the same time be expressive enough to capture all the features we want for aggressive HPC optimizations, and sufficiently restrictive to be (at least partially) statically analyzable at a reasonable cost.
- Define a module system for our medium-grain dataflow language. A program will then be divided into modules that can follow different compilation schemes and execution models but still communicate together. This will allow us to encapsulate a program that does not fit the polyhedral model into a polyhedral one and vice versa. Also, this will allow a compositional analysis and compilation, as opposed to global analysis which is limited in scalability.

3.2. Expressivity and Scalability of Static Analyses

The design and implementation of efficient compilers becomes more difficult each day, as they need to bridge the gap between *complex languages* and *complex architectures*. Application developers use languages that bring them close to the problem that they need to solve which explains the importance of high-level programming languages. However, high-level programming languages tend to become more distant from the hardware which they are meant to command.

In this research direction, we propose to design expressive and scalable static analyses for compilers. This topic is closely linked to Sections 3.1 and 3.3 since the design of an efficient intermediate representation is made while regarding the analyses it enables. The intermediate representation should be expressive enough to embed maximal information; however if the representation is too complex the design of scalable analyses will be harder.

The analyses we plan to design in this activity will of course be mainly driven by the HPC dataflow optimizations we mentioned in the preceding sections; however we will also target other kinds of analyses applicable to more general purpose programs. We will thus consider two main directions:

- Extend the applicability of the polyhedral model, in order to deal with HPC applications that do not fit totally in this category. More specifically, we plan to work on more complex control and also on complex data structures, like sparse matrices, which are heavily used in HPC.
- Design of specialized static analyses for memory diagnostic and optimization inside general purpose compilers.

For both activities, we plan to cross fertilize ideas coming from the abstract interpretation community as well as language design, dataflow semantics, and WCET estimation techniques.

Correct by construction analyses. The design of well-defined semantics for the chosen programming language and intermediate representation will allow us to show the correctness of our analyses. The precise study of the semantics of Section 3.1 will allow us to adapt the analysis to the characteristics of the language, and prove that such an adaptation is well founded. This approach will be applicable both on the source language and on the intermediate representation.

Such wellfoundedness criteria relatively to the language semantics will first be used to design our analyses, and then to study which extensions of the languages can be envisioned and analyzed safely, and which extensions (if any) are difficult to analyze and should be avoided. Here the correct identification of a core language for our formal studies (see Section 3.1) will play a crucial role as the core language should feature all the characteristics that might make the analysis difficult or incorrect.

Scalable abstract domains. We already have experience in designing low-cost semi relational abstract domains for pointers [50], [46], as well as tailoring static analyses for specialized applications in compilation [36], [54], Synchronous Dataflow scheduling [53], and extending the polyhedral model to irregular applications [21]. We also have experience in the design of various static verification techniques adapted to different programming paradigms.

3.2.1. Expected impact

The impact of this work is the significantly widened applicability of various tools/compilers related to parallelization: allow optimizations for a larger class of programs, and allow low-cost analysis that scale to very large programs.

We target both analysis for optimization and analysis to detect, or prove the absence of bugs.

3.2.2. Scientific Program

3.2.2.1. Short-term and ongoing activities.

Together with Paul Iannetta and Lionel Morel (INSA/CEA LETI), we are currently working on the *semantic rephrasing* of the polyhedral model [39]. The objective is to clearly redefine the key notions of the polyhedral model on general flowchart programs operating on arrays, lists and trees. We reformulate the algorithms that are performed to compute dependencies in a more semantic fashion, i.e. considering the program semantics instead of syntactical criteria. The next step is to express classical scheduling and code generation activities in this framework, in order to overcome the classical syntactic restrictions of the polyhedral model.

3.2.2.2. Medium-term activities.

In medium term, we want to extend the polyhedral model for more general data-structures like lists and sparse matrices. For that purpose, we need to find polyhedral (or other shapes) abstractions for non-array data-structures; the main difficulty is to deal with non-linearity and/or partial information (namely, over-approximations of the data layout, or over-approximation of the program behavior). This activity will rely on a formalization of the optimization activities (dependency computation, scheduling, compilation) in a more general Abstract-Interpretation based framework in order to make the approximations explicit.

At the same time, we plan to continue to work on scaling static analyses for general purpose programs, in the spirit of Maroua Maalej's PhD [47], whose contribution is a sequence of memory analyses inside production compilers. We already began a collaboration with Sylvain Collange (PACAP team of IRISA Laboratory) on the design of static analyses to optimize copies from the global memory of a GPU to the block kernels (to increase locality). In particular, we have the objective to design specialized analyses but with an explicit notion of cost/precision compromise, in the spirit of the paper [41] that tries to formalize the cost/precision compromise of interprocedural analyses with respect to a “context sensitivity parameter”.

3.2.2.3. Long-term activities.

In a longer-term vision, the work on scalable static analyses, whether or not directed from the dataflow activities, will be pursued in the direction of large general-purpose programs.

An ambitious challenge is to find a generic way of adapting existing (relational) abstract domains within the Single Static Information [26] framework so as to improve their scalability. With this framework, we would be able to design static analyses, in the spirit of the seminal paper [31] which gave a theoretical scheme for classical abstract interpretation analyses.

We also plan to work on the interface between the analyses and their optimization clients inside production compilers.

3.3. Compiling and Scheduling Dataflow Programs

In this part, we propose to design the compiler analyses and optimizations for the *medium-grain* dataflow model defined in section 3.1. We also propose to exploit these techniques to improve the compilation of dataflow languages based on actors. Hence our activity is split into the following parts:

- Translating a sequential program into a medium-grain dataflow model. The programmer cannot be expected to rewrite the legacy HPC code, which is usually relatively large. Hence, compiler techniques must be invented to do the translation.
- Transforming and scheduling our medium-grain dataflow model to meet some classic optimization criteria, such as throughput, local memory requirements, or I/O traffic.

- Combining agents and polyhedral kernels in dataflow languages. We propose to apply the techniques above to optimize the processes in actor-based dataflow languages and combine them with the parallelism existing in the languages.

We plan to rely extensively on the polyhedral model to define our compiler analysis. The polyhedral model was originally designed to analyze imperative programs. Analysis (such as scheduling or buffer allocation) must be redefined in light of dataflow semantics.

Translating a sequential program into a medium-grain dataflow model. The programs considered are compute-intensive parts from HPC applications, typically big HPC kernels of several hundreds of lines of C code. In particular, we expect to analyze the process code (actors) from the dataflow programs. On short ACL (Affine Control Loop) programs, direct solutions exist [56] and rely directly on array dataflow analysis [35]. On bigger ACL programs, this analysis no longer scales. We plan to address this issue by *modularizing* array dataflow analysis. Indeed, by splitting the program into processes, the complexity is mechanically reduced. This is a general observation, which was exploited in the past to compute schedules [34]. When the program is no longer ACL, a clear distinction must be made between polyhedral parts and non polyhedral parts. Hence, our medium-grain dataflow language must distinguish between polyhedral process networks, and non-polyhedral code fragments. This structure raises new challenges: How to abstract away non-polyhedral parts while keeping the polyhedrality of the dataflow program? Which trade-off(s) between precision and scalability are effective?

Medium-grain data transfers minimization. When the system consists of a single computing unit connected to a slow memory, the roofline model [59] defines the optimal ratio of computation per data transfer (*operational intensity*). The operational intensity is then translated to a partition of the computation (loop tiling) into *reuse units*: inside a reuse unit, data are transfered locally; between reuse units, data are transfered through the slow memory. On a *fine-grain* dataflow model, reuse units are exposed with loop tiling; this is the case for example in Data-aware Process Network (DPN) [23]. The following questions are however still open: How does that translate on *medium-grain* dataflow models? And fundamentally what does it mean to *tile* a dataflow model?

Combining agents and polyhedral kernels in dataflow languages. In addition to the approach developed above, we propose to explore the compilation of dataflow programming languages. In fact, among the applications targeted by the project, some of them are already thought or specified as dataflow actors (video compression, machine-learning algorithms,...).

So far, parallelization techniques for such applications have focused on taking advantage of the decomposition into agents, potentially duplicating some agents to have several instances that work on different data items in parallel [40]. In the presence of big agents, the programmer is left with the splitting (or merging) of these agents by-hand if she wants to further parallelize her program (or at least give this opportunity to the runtime, which in general only sees agents as non-malleable entities). In the presence of arrays and loop-nests, or, more generally, some kind of regularity in the agent's code, however, we believe that the programmer would benefit from automatic parallelization techniques such as those proposed in the previous paragraphs. To achieve the goal of a totally integrated approach where programmers write the applications they have in mind (application flow in agents where the agents' code express potential parallelism), and then it is up to the system (compiler, runtime) to propose adequate optimizations, we propose to build on solid formal definition of the language semantics (thus the formal specification of parallelism occurring at the agent level) to provide hierarchical solutions to the problem of compilation and scheduling of such applications.

3.3.1. Expected impact

In general, splitting a program into simpler processes simplifies the problem. This observation leads to the following points:

- By abstracting away irregular parts in processes, we expect to structure the long-term problem of handling irregular applications in the polyhedral model. The long-term impact is to widen the applicability of the polyhedral model to irregular kernels.
- Splitting a program into processes reduces the problem size. Hence, it becomes possible to scale traditionally expensive polyhedral analysis such as scheduling or tiling to quote a few.

As for the third research direction, the short term impact is the possibility to combine efficiently classical dataflow programming with compiler polyhedral-based optimizations. We will first propose ad-hoc solutions coming from our HPC application expertise, but supported by strong theoretical results that prove their correctness and their applicability in practice. In the longer term, our work will allow specifying, designing, analyzing, and compiling HPC dataflow applications in a unified way. We target semi-automatic approaches where pertinent feedback is given to the developer during the development process.

3.3.2. Scientific Program

3.3.2.1. Short-term and ongoing activities.

We are currently working on the RTM (Reverse-Time Migration) kernel for oil and gas applications ($\approx$ 500 lines of C code). This kernel is long enough to be a good starting point, and small enough to be handled by a polyhedral splitting algorithm. We figured out the possible splittings so the polyhedral analysis can scale and irregular parts can be hidden. In a first step, we plan to define splitting metrics and algorithms to optimize the usual criteria: communication volume, latency and throughput.

Together with Lionel Morel (INSA/CEA LETI), we currently work on the evaluation of the practical advantage of combining the dataflow paradigm with the polyhedral optimization framework. We empirically build a proof-of-concept tooling approach, using existing tools on existing languages [38]. We combine dataflow programming with polyhedral compilation in order to enhance program parallelization by leveraging both inter-agent parallelism and intra-agent parallelism (i.e., regarding loop nests inside agents). We evaluate the approach practically, on benchmarks coming from image transformation or neural networks, and the first results demonstrate that there is indeed a room for further improvement.

3.3.2.2. Medium-term activities.

The results of the preceding paragraph are partial and have been obtained with a simple experimental approach only using off-the-shelf tools. We are thus encouraged to pursue research on combining expertise from dataflow programming languages and polyhedral compilation. Our long term objective is to go towards a formal framework to express, compile, and run dataflow applications with intrinsic instruction or pipeline parallelism.

We plan to investigate in the following directions:

- Investigate how polyhedral analysis extends on modular dataflow programs. For instance, how to modularize polyhedral scheduling analysis on our dataflow programs?
- Develop a proof of concept and validate it on linear algebra kernels (SVD, Gram-Schmidt, etc.).
- Explore various areas of applications from classical dataflow examples, like radio and video processing, to more recent applications in deep learning algorithmic. This will enable us to identify some potential (intra and extra) agent optimization patterns that could be leveraged into new language idioms.

3.3.2.3. Long-term activities.

Current work focus on purely polyhedral applications. Irregular parts are not handled. Also, a notion of tiling is required so the communications of the dataflow program with the outside world can be tuned with respect to the local memory size. Hence, we plan to investigate the following points:

- Assess simple polyhedral/non polyhedral partitioning: How non-polyhedral parts can be hidden in processes/channels? How to abstract the dataflow dependencies between processes? What would be the impact on analyses? We target programs with irregular control (e.g., while loop, early exits) and regular data (arrays with affine accesses).
- Design tiling schemes for modular dataflow programs: What does it mean to tile a dataflow program? Which compiler algorithms to use?
- Implement a mature compiler infrastructure from the front-end to code generation for a reasonable subset of the representation.

3.4. HLS-specific Dataflow Optimizations

The compiler analyses proposed in section 3.3 do not target a specific platform. In this part, we propose to leverage these analysis to develop source-level optimizations for high-level synthesis (HLS).

High-level synthesis consists in compiling a kernel written in a high-level language (typically in C) into a circuit. As for any compiler, an HLS tool consists in a *front-end* which translates the input kernel into an *intermediate representation*. This intermediate representation captures the control/flow dependences between computation units, generally in a hierarchical fashion. Then, the *back-end* maps this intermediate representation to a circuit (e.g. FPGA configuration). We believe that HLS tools must be thought as fine-grain automatic parallelizers. In classic HLS tools, the parallelism is expressed and exploited at the back-end level during the scheduling and the resource allocation of arithmetic operations. We believe that it would be far more profitable to derive the parallelism at the front-end level.

Hence, CASH will focus on the *front-end* pass and the *intermediate representation*. Low-level *back-end* techniques are not in the scope of CASH. Specifically, CASH will leverage the dataflow representation developed in Section 3.1 and the compilation techniques developed in Section 3.3 to develop a relevant intermediate representation for HLS and the corresponding front-end compilation algorithms.

Our results will be evaluated by using existing HLS tools (e.g., Intel HLS compiler, Xilinx Vivado HLS). We will implement our compiler as a source-to-source transformation in front of HLS tools. With this approach, HLS tools are considered as a “back-end black box”. The CASH scheme is thus: (i) *front-end*: produce the CASH dataflow representation from the input C kernel. Then, (ii) turn this dataflow representation to a C program with pragmas for an HLS tool. This step must convey the characteristics of the dataflow representation found by step (i) (e.g. dataflow execution, fifo synchronisation, channel size). This source-to-source approach will allow us to get a full source-to-FPGA flow demonstrating the benefits of our tools while relying on existing tools for low-level optimizations. Step (i) will start from the DCC tool developed by Christophe Alias, which already produces a dataflow intermediate representation: the Data-aware Process Networks (DPN) [23]. Hence, the very first step is then to chose an HLS tool and to investigate which input should be fed to the HLS tool so it “respects” the parallelism and the resource allocation suggested by the DPN. From this basis, we plan to investigate the points described thereafter.

Roofline model and dataflow-level resource evaluation. Operational intensity must be tuned according to the roofline model. The roofline model [59] must be redefined in light of FPGA constraints. Indeed, the peak performance is no longer constant: it depends on the operational intensity itself. The more operational intensity we need, the more local memory we use, the less parallelization we get (since FPGA resources are limited), and finally the less performance we get! Hence, multiple iterations may be needed before reaching an efficient implementation. To accelerate the design process, we propose to iterate at the dataflow program level, which implies a fast resource evaluation at the dataflow level.

Reducing FPGA resources. Each parallel unit must use as little resources as possible to maximize parallel duplication, hence the final performance. This requires to factorize the control and the channels. Both can be achieved with source-to-source optimizations at dataflow level. The main issue with outputs from polyhedral optimization is large piecewise affine functions that require a wide silicon surface on the FPGA to be computed. Actually we do not need to compute a closed form (expression that can be evaluated in bounded time on the FPGA) *statically*. We believe that the circuit can be compacted if we allow control parts to be evaluated dynamically. Finally, though dataflow architectures are a natural candidate, adjustments are required to fit FPGA constraints (2D circuit, few memory blocks). Ideas from systolic arrays [52] can be borrowed to re-use the same piece of data multiple times, despite the limitation to regular kernels and the lack of I/O flexibility. A trade-off must be found between pure dataflow and systolic communications.

Improving circuit throughput. Since we target streaming applications, the throughput must be optimized. To achieve such an optimization, we need to address the following questions. How to derive an optimal upper bound on the throughput for polyhedral process network? Which dataflow transformations should be performed to reach it? The limiting factors are well known: I/O (decoding of burst data), communications through addressable channels, and latencies of the arithmetic operators. Finally, it is also necessary to find the right methodology to measure the throughput statically and/or dynamically.

3.4.1. Expected Impact

So far, the HLS front-end applies basic loop optimizations (unrolling, flattening, pipelining, etc.) and use a Hierarchical Control Flow Graph-like representation with data dependencies annotations (HCDFG). With this approach, we intend to demonstrate that polyhedral analysis combined with dataflow representations is an effective solution for HLS tools.

3.4.2. Scientific Program

3.4.2.1. Short-term and ongoing activities.

The HLS compiler designed in the CASH team currently extracts a fine-grain parallel intermediate representation (DPN [23], [22]) from a sequential program. We will not write a back-end that produces code for FPGA but we need to provide C programs that can be fed into existing C-to-FPGA compilers. However we obviously need an end-to-end compiler for our experiments. One of the first task of our HLS activity is to develop a DPN-to-C code generator suitable as input to an existing HLS tool like Vivado HLS. The generated code should exhibit the parallelism extracted by our compiler, and allow generating a final circuit more efficient than the one that would be generated by our target HLS tool if ran directly on the input program. Source-to-source approaches have already been experimented successfully, e.g. in Alexandru Plesco's PhD [51].

3.4.2.2. Medium-term activities.

Our DPN-to-C code generation will need to be improved in many directions. The first point is the elimination of redundancies induced by the DPN model itself: buffers are duplicated to allow parallel reads, processes are produced from statements in the same loop, hence with the same control automaton. Also, multiplexing uses affine constraints which can be factorized [24]. We plan to study how these constructs can be factorized at C-level and to design the appropriate DPN-to-C translation algorithms.

Also, we plan to explore how on-the-fly evaluation can reduce the complexity of the control. A good starting point is the control required for the load process (which fetch data from the distant memory). If we want to avoid multiple load of the same data, the FSM (Finite State Machine) that describes it is usually very complex. We believe that dynamic construction of the load set (set of data to load from the main memory) will use less silicon than an FSM with large piecewise affine functions computed statically.

3.4.2.3. Long-term activities.

The DPN-to-C compiler opens new research perspectives. We will explore the roofline model accuracy for different applications by playing on DPN parameters (tile size). Unlike the classical roofline model, the peak performance is no longer assumed to be constant, but decreasing with operational intensity [60]. Hence, we expect a *unique* optimal set of parameters. Thus, we need to build a DPN-level cost model to derive an interval containing the optimal parameters.

Also, we want to develop DPN-level analysis and transformation to quantify the optimal reachable throughput and to reach it. We expect the parallelism to increase the throughput, but in turn it may require an operational intensity beyond the optimal point discussed in the first paragraph. We will assess the trade-offs, build the cost-models, and the relevant dataflow transformations.

3.5. Simulation of Hardware

Complex systems such as systems-on-a-chip or HPC computer with FPGA accelerator comprise both hardware and software parts, tightly coupled together. In particular, the software cannot be executed without the hardware, or at least a simulator of the hardware.

Because of the increasing complexity of both software and hardware, traditional simulation techniques (Register Transfer Level, RTL) are too slow to allow full system simulation in reasonable time. New techniques such as Transaction Level Modeling (TLM) [20] in SystemC [19] have been introduced and widely adopted in the industry. Internally, SystemC uses discrete-event simulation, with efficient context-switch using cooperative scheduling. TLM abstracts away communication details, and allows modules to communicate using function calls. We are particularly interested in the loosely timed coding style where the timing of the platform is not modeled precisely, and which allows the fastest simulations. This allowed gaining several orders of magnitude of simulation speed. However, SystemC/TLM is also reaching its limits in terms of performance, in particular due to its lack of parallelism.

Work on SystemC/TLM parallel execution is both an application of other work on parallelism in the team and a tool complementary to HLS presented in Sections 3.1 (dataflow models and programs) and 3.4 (application to FPGA). Indeed, some of the parallelization techniques we develop in CASH could apply to SystemC/TLM programs. Conversely, a complete design-flow based on HLS needs fast system-level simulation: the full-system usually contains both hardware parts designed using HLS, handwritten hardware components, and software.

We also work on simulation of the DPN intermediate representation. Simulation is a very important tool to help validate and debug a complete compiler chain. Without simulation, validating the front-end of the compiler requires running the full back-end and checking the generated circuit. Simulation can avoid the execution time of the backend and provide better debugging tools.

Automatic parallelization has shown to be hard, if at all possible, on loosely timed models [28]. We focus on semi-automatic approaches where the programmer only needs to make minor modifications of programs to get significant speedups.

3.5.1. Expected Impact

The short term impact is the possibility to improve simulation speed with a reasonable additional programming effort. The amount of additional programming effort will thus be evaluated in the short term.

In the longer term, our work will allow scaling up simulations both in terms of models and execution platforms. Models are needed not only for individual Systems on a Chip, but also for sets of systems communicating together (e.g., the full model for a car which comprises several systems communicating together), and/or heterogeneous models. In terms of execution platform, we are studying both parallel and distributed simulations.

3.5.2. Scientific Program

3.5.2.1. Short-term and ongoing activities.

We started the joint PhD (with Tanguy Sassolas) of Gabriel Busnot with CEA-LIST. The research targets parallelizing SystemC heterogeneous simulations. CEA-LIST already developed SCale [57], which is very efficient to simulate parallel homogeneous platforms such as multi-core chips. However, SCale cannot currently load-balance properly the computations when the platform contains different components modeled at various levels of abstraction. Also, SCale requires manual annotations to identify accesses to shared variables. These annotations are given as address ranges in the case of a shared memory. This annotation scheme does not work when the software does non-trivial memory management (virtual memory using a memory management unit, dynamic allocation), since the address ranges cannot be known statically. We started working on the “heterogeneous” aspect of simulations with an approach allowing changing the level of details in a simulation at runtime, and started tackling the virtual and dynamic memory management problem by porting Linux on our simulation platform.

We also started working on an improved support for simulation and debugging of the DPN internal representation of our parallelizing compiler (see Section 3.3). A previous quick experiment with simulation was to generate C code that simulates parallelism with POSIX-threads. While this simulator greatly helped debug the compiler, this is limited in several ways: simulations are not deterministic, and the simulator does not scale up since it would create a very large number of threads for a non-trivial design.

We are working in two directions. The first is to provide user-friendly tools to allow graphical inspection of traces. For example, we started working on the visualization of the sequence of steps leading to a deadlock when the situation occurs, and will give hints on how to fix the problem in the compiler. The second is to use an efficient simulator to speed up the simulation. We plan to generate SystemC/TLM code from the DPN representation to benefit from the ability of SystemC to simulate a large number of processes.

3.5.2.2. Medium-term activities.

Several research teams have proposed different approaches to deal with parallelism and heterogeneity. Each approach targets a specific abstraction level and coding style. While we do not hope for a universal solution, we believe that a better coordination of different actors of the domain could lead to a better integration of solutions. We could imagine, for example, a platform with one subsystem accelerated with SCale [57] from CEA-LIST, some compute-intensive parts delegated to sc-during [49] from Matthieu Moy, and a co-simulation with external physical solvers using SystemC-MDVP [27] from LIP6. We plan to work on the convergence of approaches, ideally both through point-to-point collaborations and with a collaborative project.

A common issue with heterogeneous simulation is the level of abstraction. Physical models only simulate one scenario and require concrete input values, while TLM models are usually abstract and not aware of precise physical values. One option we would like to investigate is a way to deal with loose information, e.g. manipulate intervals of possible values instead of individual, concrete values. This would allow a simulation to be symbolic with respect to the physical values.

Obviously, works on parallel execution of simulations would benefit to simulation of data-aware process networks (DPN). Since DPN are generated, we can even tweak the generator to guarantee some properties on the generated code, which gives us more freedom on the parallelization and partitioning techniques.

3.5.2.3. Long-term activities.

In the long term, our vision is a simulation framework that will allow combining several simulators (not necessarily all SystemC-based), and allow running them in a parallel way. The Functional Mockup Interface (FMI) standard is a good basis to build upon, but the standard does not allow expressing timing and functional constraints needed for a full co-simulation to run properly.

4. Highlights of the Year

4.1. Highlights of the Year

4.1.1. Awards

In January 2019, the paper “Static Analysis Of Binary Code With Memory Indirections Using Polyhedra” resulting from a collaboration with colleagues from Lille University, has received a best paper award of the VMCAI 2019 conference.

The paper “Godot: All the Benefits of Implicit and Explicit Futures” received the distinguished artefact at ECOOP’19.

BEST PAPERS AWARDS:

[5]
C. BALLABRIGA, J. FORGET, L. GONNORD, G. LIPARI, J. RUIZ. *Static Analysis Of Binary Code With Memory Indirections Using Polyhedra*, in "VMCAI’19 - International Conference on Verification, Model Checking, and Abstract Interpretation", Cascais, Portugal, LNCS, Springer, January 2019, vol. 11388, pp. 114-135 [*DOI :* 10.1007/978-3-030-11245-5_6], https://hal.archives-ouvertes.fr/hal-01939659

[8]
A. CHARIF, G. BUSNOT, R. MAMEESH, T. SASSOLAS, N. VENTROUX. *Fast Virtual Prototyping for Embedded Computing Systems Design and Exploration*, in "RAPIDO2019 - 11th Workshop on Rapid Simulation and Performance Evaluation: Methods and Tools", Valence, Spain, January 2019, pp. 1-8 [*DOI :* 10.1145/3300189.3300192], https://hal.archives-ouvertes.fr/hal-02023805

5. New Software and Platforms

5.1. DCC

DPN C Compiler

KEYWORDS: Polyhedral compilation - Automatic parallelization - High-level synthesis

FUNCTIONAL DESCRIPTION: Dcc (Data-aware process network C compiler) analyzes a sequential regular program written in C and generates an equivalent architecture of parallel computer as a communicating process network (Data-aware Process Network, DPN). Internal communications (channels) and external communications (external memory) are automatically handled while fitting optimally the characteristics of the global memory (latency and throughput). The parallelism can be tuned. Dcc has been registered at the APP ("Agence de protection des programmes") and transferred to the XtremLogic start-up under an Inria license.

- Participants: Alexandru Plesco and Christophe Alias
- Contact: Christophe Alias

5.2. PoCo

Polyhedral Compilation Library

KEYWORDS: Polyhedral compilation - Automatic parallelization

FUNCTIONAL DESCRIPTION: PoCo (Polyhedral Compilation Library) is a compilation framework allowing to develop parallelizing compilers for regular programs. PoCo features many state-of-the-art polyhedral program analysis and a symbolic calculator on execution traces (represented as convex polyhedra). PoCo has been registered at the APP ("agence de protection des programmes") and transferred to the XtremLogic start-up under an Inria licence.

- Participant: Christophe Alias
- Contact: Christophe Alias

5.3. MPPcodegen

Source-to-source loop tiling based on MPP

KEYWORDS: Source-to-source compiler - Polyhedral compilation

FUNCTIONAL DESCRIPTION: MPPcodegen applies a monoparametric tiling to a C program enriched with pragmas specifying the tiling and the scheduling function. The tiling can be generated by any convex polyhedron and translation functions, it is not necessarily a partition. The result is a C program depending on a scaling factor (the parameter). MPPcodegen relies on the MPP mathematical library to tile the iteration sets.

- Partner: Colorado State University
- Contact: Christophe Alias
- URL: http://foobar.ens-lyon.fr/mppcodegen/

6. New Results

6.1. Dataflow-explicit futures

Participants: Ludovic Henrio, Matthieu Moy, Amaury Maillé.

A future is a place-holder for a value being computed, and we generally say that a future is resolved when the associated value is computed. In existing languages futures are either implicit, if there is no syntactic or typing distinction between futures and non-future values, or explicit when futures are typed by a parametric type and dedicated functions exist for manipulating futures. We defined a new form of future, named data-flow explicit futures [43], with specific typing rules that do not use classical parametric types. The new futures allow at the same time code reuse and the possibility for recursive functions to return futures like with implicit futures, and let the programmer declare which values are futures and where synchronisation occurs, like with explicit futures. We prove that the obtained programming model is as expressive as implicit futures but exhibits a different behaviour compared to explicit futures. The current status of this work is the following:

- With collaborators from University of Uppsala and University of Oslo we worked on the design of programming constructs mixing implicit and dataflow-explicit futures (DeF). This has been published in ECOOP 2019 [10].
- Amaury Maillé did his internship in the Cash team (advised by Matthieu Moy and Ludovic Henrio), he worked on an implementation of DeF in the Encore language. This raised a difficulty regarding the interaction of DeF with generic types that has been partially solved. Now we need to generalize our approach to completely solve the issue.

6.2. Distributed futures

Participant: Ludovic Henrio.

We proposed the definition of *distributed futures*, a construct that provides at the same time a data container similar to a distributed vector, and a single synchronization entity that behaves similarly to a standard future. This simple construct makes it easy to program a composition, in a task-parallel way, of several massively data-parallel tasks. This work will be presented in Sac 2020 (we are currently working on the final version of the paper). This work is realised in collaboration with Pierre Leca and Wijnand Suijlen (Huawei Technologies), and Françoise Baude (Université Côte d'Azur, CNRS, I3S).

6.3. Locally abstract globally concrete semantics

Participant: Ludovic Henrio.

This research direction aims at designing a new way to write semantics for concurrent languages. The objective is to design semantics in a compositional way, where each primitive has a local behavior, and to adopt a style much closer to verification frameworks so that the design of an automatic verifier for the language is easier. The local semantics is expressed in a symbolic and abstract way, a global semantics gathers the abstract local traces and concretizes them. We have a reliable basis for the semantics of a simple language (a concurrent while language) and for a complex one (ABS), but the exact semantics and the methodology for writing it is still under development. After 2 meetings in 2019, A journal article is still being written but the visit of Reiner Hähnle in the Cash team during two months (as invited professor) in Spring 2019 should allow us to make faster progress on the topic.

This is a joint with Reiner Hähnle (TU Darmstadt), Einar Broch Johnsen, Crystal Chang Din, Lizeth Tapia Tarifa (Univ Oslo), Ka I Pun (Univ Oslo and Univ of applied science).

6.4. Memory consistency for heterogeneous systems

Participant: Ludovic Henrio.

Together with Christoph Kessler (Linköping University), we worked on the formalization of the cache coherency mechanism used in the VectorPU library developed at Linköping University. Running a program on disjoint memory spaces requires to address memory consistency issues and to perform transfers so that the program always accesses the right data. Several approaches exist to ensure the consistency of the memory accessed, we are interested here in the verification of a declarative approach where each component of a computation is annotated with an access mode declaring which part of the memory is read or written by the component. The programming framework uses the component annotations to guarantee the validity of the memory accesses. This is the mechanism used in VectorPU, a C++ library for programming CPU-GPU heterogeneous systems and this article proves the correctness of the software cache-coherence mechanism used in the library. Beyond the scope of VectorPU, this article can be considered as a simple and effective formalisation of memory consistency mechanisms based on the explicit declaration of the effect of each component on each memory space. This year, we have the following new results:

- we extended the work to support the manipulation of overlapping array. This was accepted as an extended version of our conference paper (presented at 4PAD 2018). It will be published in the JLAMP journal in 2020 [3].

6.5. PNets: Parametrized networks of automata

Participant: Ludovic Henrio.

pNets (parameterised networks of synchronised automata) are semantic objects for defining the semantics of composition operators and parallel systems. We have used pNets for the behavioral specification and verification of distributed components, and proved that open pNets (i.e. pNets with holes) were a good formalism to reason on operators and parameterized systems. This year, we have the following new results:

- A weak bisimulation theory for open pNets. This work is realized with Eric Madelaine (Inria Sophia-Antipolis) and Rabéa Ameur Boulifa (Telecom ParisTech). A journal article has been written and will be submitted in January 2020.
- A translation from BIP model to open pNets has being formalized and encoded, this work is done in collaboration with Simon Bliudze (Inria Lille). More precisely, we extend the theory of architectures developed previously for the BIP framework with the elements necessary for handling data: definition and operations on data domains, syntax and semantics of composition operators involving data transfer. To verify that individual architectures do enforce their associated properties , we provide an encoding into open pNets, an intermediate model that supports SMT-based verification. This work has been published in Coordination 2019 [6].

These works are under progress and should be continued in 2020.

6.6. Decidability results on the verification of phaser programs

Participant: Ludovic Henrio.

Together with Ahmed Rezine and Zeinab Ganjei (Linköping University) we investigated the possibility to analyze programs with phasers (a construct for synchronizing processes that generalizes locks, barrier, and publish-subscribe patterns). They work with signal and wait messages from the processes (comparing the number of wait and signal received to synchronize the processes). We proved that in many conditions, if the number of phasers or processes cannot be bounded, or if the difference between the number of signal and the number of wait signal is unbounded, then many reachability problems are undecidable. We also proposed fragments where these problems become decidable, and proposed an analysis algorithm in these cases. The results have been published in TACAS 2019 [11].

6.7. A Survey on Verified Reconfiguration

Participant: Ludovic Henrio.

We are conducting a survey on the use of formal methods to ensure safety of reconfiguration of distributed system, that is to say the runtime adaptation of a deployed distributed software system. The survey article is written together with Hélène Coullon and Simon Robillard (IMT Atlantique, Inria, LS2N, UBL), and Frédéric Loulergue (Northern Arizona University). Hélène Coullon is the coordinator and we expect the article to be submitted in 2020.

6.8. A Survey on Parallelism and Determinacy

Participants: Ludovic Henrio, Laure Gonnord, Matthieu Moy, Christophe Alias.

We have started to investigate on the solutions that exist to ensure complete or partial determinacy in parallel programs. The objective of this work is to provide a survey based on the different kinds of solutions that exist to ensure determinism or at least limit data-races in concurrent execution of programs. The study will cover language-based, compilation-based and also runtime-based solutions. We started the bibliographic studies in 2019. The objective of this work is to write and submit a survey article in 2020.

This work, coordinated by Laure Gonnord and Ludovic Henrio, also involves contributors outside the CASH team. For the moment Gabriel Radanne (Inria Paris) and Lionel Morel (CEA).

6.9. Pipeline-aware Scheduling of Polyhedral Process Networks

Participants: Christophe Alias, Julien Rudeau.

The polyhedral model is a well known framework to develop accurate and optimal automatic parallelizers for high-performance computing kernels. It is progressively migrating to high-level synthesis through polyhedral process networks (PPN), a dataflow model of computation which serves as intermediate representation for high-level synthesis. Many locks must be overcome before having a fully working polyhedral HLS tool, both from a front-end (C $\rightarrow$ PPN) and back-end (PPN $\rightarrow$ FPGA) perspective. In this work [15], we propose a front-end scheduling algorithm which reorganizes the computation of processes to maximize the pipeline efficiency of the processes' arithmetic operators. We show that our approach improve significantly the overall latency as well as the pipeline efficiency.

6.10. A Compiler Algorithm to Guide Runtime Scheduling

Participants: Christophe Alias, Samuel Thibault, Laure Gonnord.

Task-level parallelism is usually exploited by a runtime scheduler, after tasks are mapped to processing units by a compiler. In this report, we propose a compilation-centric runtime scheduling strategy. We propose a complete compilation algorithm to split the tasks in three parts, whose properties are intended to help the scheduler to take the right decisions [16]. In particular, we show how the polyhedral model may provide a precious help to compute tricky scheduling and parallelism informations. Our compiler is available and may be tried online at http://foobar.ens-lyon.fr/kut.

This is a joint work with University of Bordeaux, which will be continued next year.

6.11. fkcc: the Farkas Calculator

Participant: Christophe Alias.

We propose a new domain-specific language and a tool, FKCC, to prototype program analyses and transformations exploiting the affine form of Farkas lemma. Our language is general enough to prototype in a few lines sophisticated termination and scheduling algorithms. The tool is freely available and may be tried online via a web interface. We believe that FKCC is the missing chain to accelerate the development of program analyses and transformations exploiting the affine form of Farkas lemma.

This work has been presented in the TAPAS'19 workshop [13] and will be presented at the IMPACT'20 workshop [13].

6.12. Standard-compliant Parallel SystemC simulation of Loosely-Timed Transaction Level Models

Participant: Matthieu Moy.

To face the growing complexity of System-on-Chips (SoCs) and their tight time-tomarket constraints, Virtual Prototyping (VP) tools based on SystemC/TLM must get faster while keeping accuracy. However, the Accellera SystemC reference implementation remains sequential and cannot leverage the multiple cores of modern workstations. In this paper, we present a new implementation of a parallel and standard-compliant SystemC kernel, reaching unprecedented performances. By coupling a parallel SystemC kernel and memory access monitoring, we are able to keep SystemC atomic thread evaluation while leveraging the available host cores. Evaluations show a ×19 speed-up compared to the Accellera SystemC kernel using 33 host cores reaching speeds above 2000 Million simulated Instructions Per Second (MIPS).

This work will be published at the ASP-DAC 2020 conference.

6.13. Response time analysis of dataflow applications on a many-core processor with shared-memory and network-on-chip

Participant: Matthieu Moy.

We consider hard real-time applications running on many-core processor containing several clusters of cores linked by a Network-on-Chip (NoC). Communications are done via shared memory within a cluster and through the NoC for inter-cluster communication. We adopt the time-triggered paradigm, which is well-suited for hard real-time applications, and we consider data-flow applications, where communications are explicit.

We extend the AER (Acquisition/Execution/Restitution) execution model to account for all delays and interferences linked to communications, including the interference between the NoC interface and the memory. Indeed, for NoC communications, data is first read from the initiator's local memory, then sent over the NoC, and finally written to the local memory of the target cluster. Read and write accesses to transfer data between local memories may interfere with shared-memory communication inside a cluster, and, as far as we know, previous work did not take these interferences into account.

Building on previous work on deterministic network calculus and shared memory interference analysis, our method computes a static, time-triggered schedule for an application mapped on several clusters. This schedule guarantees that deadlines are met, and therefore provides a safe upper bound to the global worst-case response time.

This work was published at RTNS 2019 [14].

6.14. Smart placement of dynamically allocated objects for heterogeneous memory

Participant: Matthieu Moy.

As part of a partnership with the CITI laboratory (Tristan Delizy's PhD, co-supervised with Guillaume Salagnac and Tanguy Risset), we worked on dynamic memory memory allocation for embedded systems with heterogeneous memory. Unlike cache-based systems, our target architecture exposes several memory banks with different performance characteristics directly to the software, without any hardware mechanism like a cache or an MMU for memory management. The software needs to chose which memory bank to use at allocation time, and cannot change this choice afterwards. We proposed a profiling-based placement policy that is shown to be near-optimal for several applications, and performs much better than naive placement policies especially for systems with a small fraction of fast memory.

This work documented as part of Tristan Delizy's Ph.D manuscript, and we plan to submit it for a journal publication in 2020.

6.15. Static Analysis Of Binary Code With Memory Indirections Using Polyhedra

Participant: Laure Gonnord.

Together with Clement Ballabriga, Julien Forget, Giuseppe Lipari, and Jordy Ruiz (University of Lille), we proposed in 2018 a new abstract domain for static analysis of binary code. Our motivation stems from the need to improve the precision of the estimation of the Worst-Case Execution Time (WCET) of safety-critical real-time code. WCET estimation requires computing information such as upper bounds on the number of loop iterations, unfeasible execution paths, etc. These estimations are usually performed on binary code, mainly to avoid making assumptions on how the compiler works. Our abstract domain, based on polyhedra and on two mapping functions that associate polyhedra variables with registers and memory, targets the precise computation of such information. We prove the correctness of the method, and demonstrate its effectiveness on benchmarks and examples from typical embedded code.

The results have been presented to VMCAI'19 on Model Checking and Abstract Interpretation [5] and has received the best paper award of the conference.

6.16. Polyhedral Value Analysis as Fast Abstract Interpretation

Participant: Laure Gonnord.

Together with Tobias Grosser, (ETH Zurich, Switzerland), Siddhart Bhat, (IIIT Hydrabad, India), Marcin Copik (ETH Zurich, Switzerland), Sven Verdoolaege (Polly Labs, Belgium) and Torsten Hoefler (ETH Zurich, Switzerland), we tried to bridge the gap between the well founded classical abstract interpretation techniques and their usage in production compilers.

We formulate the polyhedral value analysis (a classical algorithm in production compilers like LLVM, scalar evolution based on Presburger set as abstract interpretation), and rephrase a complete value and validity

In 2019, the formalisation has been rephrased in a simplier way and extented to deal with more llvm-related semantics (undefined behavior, poisoned values) and we started a collaboration with David Monniaux, Verimag, on this topic.

The paper is being rewritten and we are also writing a project on which we would extend our method to mode complex polyhedral transformations in a context of formally verified tools.

6.17. Decision results for solving Horn Clauses with arrays

Participants: Laure Gonnord, Julien Braine.

Many approaches exist for verifying programs operating on Boolean and integer values (e.g. abstract interpretation, counterexample-guided abstraction refinement using interpolants), but transposing them to array properties has been fraught with difficulties. In the context of the Phd of Julien Braine, we propose to work directly on horn clauses, because we think that it is a suitable intermediate representation for verifying programs.

Currently, two techniques strike out to infer very precise quantified invariants on arrays using Horn clauses: a quantifier instantiation method [1] and a cell abstraction method that can be rephrased on Horn clauses. However, the quantifier instantiation method is parametrized by an heuristic and finding a good heuristic is a major challenge, and the cell abstraction method uses an abstract interpretation to completely remove arrays and is limited to linear Horn clauses. We combine these two techniques. We provide an heuristic for the quantifier instantiation method of [29] by using the ideas from the cell abstraction method of [48] and discover a requirement such that, when met, the heuristic is complete, that is, there is no loss of information by using that heuristic. Furthermore, we prove that Horn clauses that come from program semantic translation verify the requirement and therefore, we have an optimal instantiation technique for program analysis.

This work is done in collaboration with David Monniaux (Verimag), coadvisor of the PhD of Julien Braine. A journal paper is currently being written for submission early 2020.

6.18. Scheduling Trees

Participants: Laure Gonnord, Paul Iannetta.

As a first step to schedule non polyhedral computation kernels, we investigated the tree datastructure. A large bibliography on tree algorithmics and complexity leds us to chose to work on balanced binary trees, for which we have designed algorithms to change their memory layout into adjacent arrays. We rephrased the classical algorithms (construction, search, destruction ...) in this setting, and implemented them in C.

The conclusion of this study is unfortunately negative : the locality gain in transforming trees into linear structures is not contrabalanced by a better cache usage, all our codes have been slowed down in the process. Our experiments are still in progress, but our hypothesis is that our trees are too sparse to be more clever that the *malloc* implementation.

A research paper will be published early 2020. This work is done in collaboration with Lionel Morel (CEA Grenoble), coadvisor of the PhD of Paul Iannetta.

6.19. Formalisation of the Polyhedral Model

Participants: Laure Gonnord, Paul Iannetta.

Last year, together with Lionel Morel (Insa/CEA) and Tomofumi Yuki (Inria, Rennes), we revisited the polyhedral model's key analysis, dependency analysis, published in a research report [44]. This year we pursued in this direction. We have now a better formalisation, and a better understanding of the expressivity and applicability.

We still have one step to study in order to be able to have a full semantic polyhedral model: properly formalise code scheduling and code generation within our semantic model.

This work is made in collaboration with Lionel Morel (CEA Grenoble) who coadvise Paul Iannetta.

6.20. Semantics diffs in LLVM

Participants: Laure Gonnord, Matthieu Moy.

Laure Gonnord and Matthieu Moy have coadvised a Master research Project ("TER") early in 2019 , whose objective was to study the LLVM LLVM compiler infrastructure with software engineering techniques in order to characterise how sequences of code analyses and transformations behave. The project has lead to a sequence of tools to evaluate experimentally how a sequence of passes influence performance.

Laure Gonnord and Matthieu Moy have, together with Sebastien Mosser, coadvised a second internship at UQAM for three months, between May and July 2019. During his internship, Sebastien Michelland has demonstrated that textual diffs are not sufficient to fully characterise the behaviours of code transformation inside compilers. He analysed llvm-diff, a tool of the distribution that makes an analysis at the intermediate representation level, and gives first hints to define a proper notion of semantic diff for this application.

For these interships two research reports have been produced.

This work was done in the context of an ongoing collaboration with Sebastien Mosser, previously in Nice, and now at UQAM. An Inria associate team was proposed for 2020-2023 on similar topics.

7. Bilateral Contracts and Grants with Industry

7.1. Bilateral Contracts with Industry

CIFRE Ph.D of Julien Emmanuel with Bull/Atos, hosted by Inria. 2020-2023.

8. Partnerships and Cooperations

8.1. National Initiatives

8.1.1. ANR

- Laure Gonnord's "Jeune Chercheur" ANR, CODAS, has started in January 2018 (42 months).

8.1.2. Scientific Advising

- Christophe Alias is scientific advisor (concours scientifique, 20%) for the XTREMLOGIC start-up.

8.2. International Initiatives

8.2.1. Informal International Partners

- Laure Gonnord has regular collaborations with Fernando Pereira from UFMG, Brasil (5 publications in total, last in 2017). End of 2019 they have restarted discussions with Gabriel Radanne about proving termination properties of linux kernel BPF programs. These programs must be always terminating, and we hope to be able to prove these properties in a scalable way with the termite analyser.
- In 2018 Laure Gonnord has began a collaboration with Tobias Grösser, from ETH Zurich, and in end of 2019 this collaboration has been extended to involved more people of Verimag (David Monniaux) and CASH, in the contexte of a europeean project proposal around certified polyhedral optimisation.
- In 2019, Laure Gonnord has pursued her collaboration with Sebastien Mosser, who moved from univ Nice to UQAM (Quebec, Canada). This collaboration has led to shared interns and a "inria associate team" proposal late in october 2019, which got accepted in January 2019.
- Ludovic Henrio has regular collaborations with: University of Oslo and University of Bergen in Norway (Cristal C. Din, Einar B. Johnsen, and Silvia Lizeth. Tapia Tarifa, Violet K.I. Pun); Reiner Hähnle (TU Darmstadt), Wolfgang Ahrendt (Chalmers); Kiko Fernandez-Reyes, Dave Clarke, and Tobiaas Wrigstad (Univ Uppsala); Christoph Kessler and Ahmed Rezine (Univ of Linköping).

8.3. International Research Visitors

8.3.1. Visits of International Scientists

8.3.1.1. Internships

- Amaury Maillé, M2: from Dec 2018 to Aug 2019 (6 months in total), "Dataflow explicit futures: Formalisation and/or experimentation".
- Julien Rudeau, INSA 4A, from to Apr 2019 to Aug 2019, "Ordonnancement sous contrainte de pipeline", supervised by Christophe Alias.
- Julien Philippon, EPITECH 1A, from to Jul 2019 to Dec 2019, "Compiling dataflow models to circuits", supervised by Christophe Alias and Matthieu Moy.
- Mohamed Hadjoudj, ENS Paris-Saclay 1A, from Jun 2019 to Jul 2019, "Parallélisation sous contrainte de ressources", supervised by Christophe Alias.
- Julian Bruyat, Lyon 1 M1, part-time from January 2019 to May 2019, "Outillage pour l'étude de l'impact de l'ordre des passes de LLVM", supervised by Laure Gonnord and Matthieu Moy.
- Sebastien Michelland, ENS de Lyon M1, abroad co-supervision by Laure Gonnord and Matthieu Moy with main supervision Sebastien Mosser at UQAM (Canada), from May 2019 to July 2019 "Exploration et cartographie des passes de LLVM".

9. Dissemination

9.1. Promoting Scientific Activities

9.1.1. Scientific Events: Organisation

9.1.1.1. Member of the Organizing Committees

- Laure Gonnord animates the french compilation community since 2010 (http://compilfr.ens-lyon.fr.

9.1.2. Scientific Events: Selection

9.1.2.1. Chair of Conference Program Committees

- Laure Gonnord was PC chair and organizer of the 8th Numerical and Symbolic Abstract Domain (NSAD, satellites workshop of SAS 2019, FM2019, in Porto.)
- Ludovic Henrio was chair of the ICE'19 workshop.

9.1.2.2. Member of the Conference Program Committees

- Laure Gonnord is a PC member of **CAV** 2019 Conference on Computer-Aided Verification. She will be PC member of SBLP'20 (Brazilian Symposium on Programming Languages).
- Ludovic Henrio has been a PC member of FASE 2019, ACSD 2019, Coordination 2019, HLPP 2019, AGERE 2019, and iFM 2019
- Christophe Alias has been a PC member of COMPAS'19.

9.1.2.3. Reviewer

- Christophe Alias was reviewer for MCSoC'19.
- Matthieu Moy was reviewer for CAV'19, IFM'19, MCSOC'19

9.1.3. Journal

9.1.3.1. Reviewer - Reviewing Activities

- Ludovic Henrio was reviewer for the HLPP'19 special issue in IJPP. He is also guest editor for the special issues associated to the ICE workshop of the last 4 years.
- Christophe Alias was reviewer for ACM TOPC.
- Matthieu Moy was reviewer for Journal of Electronics.

9.1.4. Invited Talks

- Matthieu Moy presented a talk "La génération de code temps réel sur architecture many-coeur" at the annual colloquium of GDR SOC2.

9.1.5. Research Administration

- Laure Gonnord is elected member of the LIP council and the Fédération d'Informatique de Lyon council.
- Laure Gonnord is elected member of the Inria National "Commission d'Evaluation" from september 2019.

9.2. Teaching - Supervision - Juries

9.2.1. Teaching

Licence:

- Christophe Alias, Compilation, CM+TD, 27h, 3A, INSA Centre Val de Loire.
- Laure Gonnord, Algorithmic, C++ Programming, TD+TP, 42h, L2, UCBL
- Laure Gonnord, Operating Systems, TD+TP, 26h, L2, UCBL

- Matthieu Moy, Concurrent Programming, CM+TD+TP, 57h, L3, UCBL.
- Matthieu Moy, Recursive Programming, TD+TP, 48h, L1, UCBL.
- Matthieu Moy, Git, CM+TP: 12h, L3, UCBL.
- Amaury Maillé, Concurrent Programming, 8h TD, 14 h 30 TP, L3, UCBL.
- Paul Iannetta, ACM, TD, 32h, L3, ENS de Lyon.
- Paul Iannetta, Projet 1, TD, 36h, L3, ENS de Lyon.
- Paul Iannetta, Colloquium L3, TD, 2h, L3, ENS de Lyon
- Paul Iannetta, Jury de stage, TD, 4h, L3, ENS de Lyon
- Julien Braine, ASR, TP et TD, 32h, L3, ENS de Lyon.

Master:

- Christophe Alias, Compiler optimizations for embedded applications, CM+TD, 27h, 4A, INSA Centre Val de Loire.
- Laure Gonnord, Compilation and Program Analysis, CM, 10h, TP 8h, M1, ENS de Lyon.
- Laure Gonnord, Compilation and program transformations, CM+TD+TP, 35h, M1, UCBL.
- Laure Gonnord, Real Time Systems, CM+TD+TP, 30h, M1, UCBL.
- Laure Gonnord, Distributed Systems, TD 9h, M1, UCBL.
- Laure Gonnord, Graphs, Complexity, Algorithmics, M1 MEEF (CAPES Maths, prepa), CM+TD+TP+oral training, 18h, UCBL.
- Laure Gonnord, Algorithmics, Systems M1 MEEF (CAPES NSI, prepa), CM+TD, 15h, UCBL.
- Laure Gonnord, Algorithmics, Architecture, Systems, DIU EIL, CM+TD+TP, 60h, UCBL.
- Matthieu Moy, Software Engineering, CM+TD+TP, 25h, M1, UCBL.
- Matthieu Moy, Compilation and Program Analysis, TP, 16h, M1, ENS de Lyon.
- Matthieu Moy, Compilation and program transformations, TD+TP, 25h, M1, UCBL.
- Ludovic Henrio, Compilation and Program Analysis, CM, 10h; TP, 6h, M1, ENS de Lyon.
- Ludovic Henrio, Distributed Systems: an algorithmic approach, CM+TD, 3h, M2 Specialite IFI (Ingénierie et Fondements de l'Informatique), parcours CSSR, and UBINET, Université de Nice Sophia-Antipolis.
- Amaury Maillé, Networks, 2h TD, M1, UCBL.
- Paul Iannetta, Projet Intégré, TD, 14h, M1, ENS de Lyon.
- Paul Iannetta, Jury de stage, TD, 4h, M1, ENS de Lyon
- Julien Braine, APPD, TP et TD, 28h, M1, ENS de Lyon.

9.2.2. Supervision

- PhD in progress: Gabriel Busnot, "Accélération SystemC pour la co-simulation multi-physique et la simulation de modèles hétérogènes en complexité", Univ. Lyon 1, started in october 2017, supervised by Matthieu Moy (LIP) and Tanguy Sassolas (CEA-LIST).
- PhD: Tristan Delizy, "Dynamic Memory Management For Embedded Non-Volatile Memory", INSA Lyon, started in October 2016, supervised by Guillaume Salagnac (CITI), Tanguy Risset (CITI), Kevin Marquet (CITI) and Matthieu Moy (LIP).
- PhD in progress (from Sept. 2018): Paul Iannetta "Complex data structures scheduling for optimizing compilers", supervised by Lionel Morel (CITI/CEA) and Laure Gonnord (LIP).

- PhD in progress (from Sept. 2018): Julien Braine "Horn Clauses as an Efficient Intermediate Representation for Data Structure Verification", supervised by David Monniaux (CNRS/Verimag) and Laure Gonnord (LIP).
- PhD in progress: Pierre Leca, "Distributed BSP: Active Objects for BSPlib programs", CIFRE Huawei/UNS, started in August 2017, supervised by Gaëtan Hains (Huawei), Wijnand Suijlen (Huawei), Françoise Baude (UNS./I3S), Ludovic Henrio (LIP).
- PhD in progress: Amaury Maillé, "Programming model to assemble compute kernels safely and efficiently: Future- based synchronization for arrays and matrices", ENS Lyon, supervised by Matthieu Moy and Ludovic Henrio. Started in October 2019, supervised by Gaëtan Hains (Huawei), Wijnand Suijlen (Huawei), Françoise Baude (UNS./I3S), Ludovic Henrio (LIP).

9.2.3. Juries

- Laure Gonnord was external jury member for the PhD of Yohan Uguen, "High-level synthesis and arithmetic optimizations" (INSA de Lyon).
- Ludovic Henrio was part of the reading commitee of Keyvan Azadbakht (Universiteit Leiden).
- Christophe Alias was a reviewer and member of the PhD committee of Hang YU (Université Grenoble-Alpes).
- Christophe Alias was *correcteur* and jury member for *concours d'admission X/ENS*.
- Laure Gonnord was jury member for the *C*oncours d'admission de l'Agrégation de Sciences Industrielles, spécialité Informatique Industrielle, in June 2019.
- Matthieu Moy was reviewer for the Ph.D of Hamza Deroui, "Étude et implantation d'algorithmes pour l'ordonnancement d'applications dataflow" (INSA Rennes).

9.3. Popularization

9.3.1. Articles and contents

- "Pourquoi créer des nouveaux langages de programmation?" Ludovic Henrio. Interstices. Janvier 2019.

9.3.2. Education

- Laure Gonnord has participated to the construction of the diploma "teaching computer science in high schools" in Lyon, and also the new preparatory cursus for the "CAPES-NSI" at Lyon1 university.

9.3.3. Interventions

- Laure Gonnord was invited by the "Association des profs de Prepa" to give her expertise about teaching computer science in the first universitary cycle. The context is the creation of a new preparatory class with a computer science minor "MPI" (Maths, Physics, Computer Science).
- Matthieu Moy presented a talk "Se lancer dans le logiciel libre quand on est étudiant (ou pas)" at "Campus du libre", Lyon.

10. Bibliography

Publications of the year

Doctoral Dissertations and Habilitation Theses

[1] C. ALIAS. *Contributions to Program Optimization and High-Level Synthesis*, ENS de Lyon, May 2019, Habilitation à diriger des recherches, https://hal.inria.fr/tel-02151877

Articles in International Peer-Reviewed Journals

[2] W. AHRENDT, L. HENRIO, W. OORTWIJN. *Who is to Blame? Runtime Verification of Distributed Objects with Active Monitors*, in "Electronic Proceedings in Theoretical Computer Science", August 2019, vol. 302, pp. 32-46, https://arxiv.org/abs/1908.10042 - In Proceedings VORTEX 2018, arXiv:1908.09302 [*DOI :* 10.4204/EPTCS.302.3], https://hal.archives-ouvertes.fr/hal-02303148

[3] L. HENRIO, C. KESSLER, L. LI. *Leveraging access mode declarations in a model for memory consistency in heterogeneous systems*, in "Journal of Logical and Algebraic Methods in Programming", January 2020, vol. 110, pp. 1-17, 100498, https://arxiv.org/abs/1910.11110 [*DOI :* 10.1016/J.JLAMP.2019.100498], https://hal.archives-ouvertes.fr/hal-02331964

International Conferences with Proceedings

[4] C. ALIAS. *fkcc: the Farkas Calculator*, in "10th Workshop on Tools for Automatic Program Analysis", Porto, Portugal, Lecture Notes in Computer Science, Springer, December 2019, https://hal.inria.fr/hal-02414224

[5] *Best Paper*
C. BALLABRIGA, J. FORGET, L. GONNORD, G. LIPARI, J. RUIZ. *Static Analysis Of Binary Code With Memory Indirections Using Polyhedra*, in "VMCAI'19 - International Conference on Verification, Model Checking, and Abstract Interpretation", Cascais, Portugal, LNCS, Springer, January 2019, vol. 11388, pp. 114-135 [*DOI :* 10.1007/978-3-030-11245-5_6], https://hal.archives-ouvertes.fr/hal-01939659.

[6] S. BLIUDZE, L. HENRIO, E. MADELAINE. *Verification of concurrent design patterns with data*, in "21th International Conference on Coordination Languages and Models (COORDINATION)", Kongens Lyngby, Denmark, H. R. NIELSON, E. TUOSTO (editors), Coordination Models and Languages, Springer International Publishing, 2019, vol. LNCS-11533, pp. 161-181, t Part 4: Coordination Patterns [*DOI :* 10.1007/978-3-030-22397-7_10], https://hal.archives-ouvertes.fr/hal-02143782

[7] G. BUSNOT, T. SASSOLAS, N. VENTROUX, M. MOY. *Standard-compliant Parallel SystemC simulation of Loosely-Timed Transaction Level Models*, in "25th Asia and South Pacific Design Automation Conference (ASP-DAC 2020)", Beijing, China, January 2020, https://hal.archives-ouvertes.fr/hal-02416253

[8] *Best Paper*
A. CHARIF, G. BUSNOT, R. MAMEESH, T. SASSOLAS, N. VENTROUX. *Fast Virtual Prototyping for Embedded Computing Systems Design and Exploration*, in "RAPIDO2019 - 11th Workshop on Rapid Simulation and Performance Evaluation: Methods and Tools", Valence, Spain, January 2019, pp. 1-8 [*DOI :* 10.1145/3300189.3300192], https://hal.archives-ouvertes.fr/hal-02023805.

[9] M. DUPONT DE DINECHIN, M. SCHUH, M. MOY, C. MAÏZA. *Scaling Up the Memory Interference Analysis for Hard Real-Time Many-Core Systems*, in "Design, Automation and Test in Europe Conference (DATE)", Grenoble, France, March 2020, https://hal.archives-ouvertes.fr/hal-02431273

[10] K. FERNANDEZ-REYES, D. CLARKE, L. HENRIO, E. BROCH JOHNSEN, T. WRIGSTAD. *Godot: All the Benefits of Implicit and Explicit Futures*, in "ECOOP 2019 - 33rd European Conference on Object-Oriented Programming", London, Royume-Uni, Leibniz International Proceedings in Informatics (LIPIcs), 2019, pp. 1-28, https://hal.archives-ouvertes.fr/hal-02302214

[11] Z. GANJEI, A. REZINE, L. HENRIO, P. ELES, Z. PENG. *On Reachability in Parameterized Phaser Programs*, in "TACAS 2019 - 25th International Conference on Tools and Algorithms for the Construction and Analysis of Systems", Prague, Czech Republic, LNCS, Springer, April 2019, vol. 11427, pp. 299-315, https://arxiv.org/abs/1811.07142 [*DOI :* 10.1007/978-3-030-17462-0_17], https://hal.archives-ouvertes.fr/hal-02061520

[12] P. LECA, L. HENRIO, F. BAUDE, W. SUIJLEN. *Distributed futures for efficient data transfer between parallel processes*, in "The 35th ACM/SIGAPP Symposium On Applied Computing", Brno, Czech Republic, March 2020 [*DOI :* 10.1145/3341105.3373932], https://hal.archives-ouvertes.fr/hal-02417953

Conferences without Proceedings

[13] C. ALIAS. *Farkas Lemma made easy*, in "IMPACT 2020 - 10th International Workshop on Polyhedral Compilation Techniques", Bologna, Italy, December 2019, pp. 1-6, https://hal.inria.fr/hal-02422033

[14] A. GRAILLAT, C. MAIZA, M. MOY, P. RAYMOND, B. DUPONT DE DINECHIN. *Response Time Analysis of Dataflow Applications on a Many-Core Processor with Shared-Memory and Network-on-Chip*, in "RTNS 2019 - 27th International Conference on Real-Time Networks and Systems", Toulouse, France, ACM, November 2019, pp. 61-69 [*DOI :* 10.1145/3356401.3356416], https://hal.archives-ouvertes.fr/hal-02320463

Research Reports

[15] C. ALIAS, J. RUDEAU. *Pipeline-aware Scheduling of Polyhedral Process Networks*, Inria Grenoble - Rhone-Alpes, December 2019, n^{o} RR-9314, https://hal.inria.fr/hal-02414340

[16] C. ALIAS, S. THIBAULT, L. GONNORD. *A Compiler Algorithm to Guide Runtime Scheduling*, Inria Grenoble ; Inria Bordeaux - Sud-Ouest, December 2019, n^{o} RR-9315, https://hal.inria.fr/hal-02421327

Scientific Popularization

[17] L. HENRIO. *Pourquoi créer des nouveaux langages de programmation ?*, in "Interstices", January 2019, https://hal.inria.fr/hal-02008111

Other Publications

[18] C. ALIAS. *On Channel Restructuring for Complete FIFO Recovery*, November 2019, ICCD 2019 - 37th IEEE International Conference on Computer Design, Poster, https://hal.inria.fr/hal-02433318

References in notes

[19] *IEEE 1666 Standard: SystemC Language Reference Manual*, Open SystemC Initiative, 2011, http://www.accellera.org/

[20] *OSCI TLM-2.0 Language Reference Manual*, Open SystemC Initiative (OSCI), June 2008, http://www.accellera.org/downloads/standards

[21] C. ALIAS, A. DARTE, P. FEAUTRIER, L. GONNORD. *Multi-dimensional Rankings, Program Termination, and Complexity Bounds of Flowchart Programs*, in "International Static Analysis Symposium (SAS'10)", 2010

[22] C. ALIAS, A. PLESCO. *Method of Automatic Synthesis of Circuits, Device and Computer Program associated therewith*, April 2014, Patent FR1453308

[23] C. ALIAS, A. PLESCO. *Data-aware Process Networks*, Inria - Research Centre Grenoble – Rhône-Alpes, June 2015, n^o RR-8735, 32 p. , https://hal.inria.fr/hal-01158726

[24] C. ALIAS, A. PLESCO. *Optimizing Affine Control with Semantic Factorizations*, in "ACM Transactions on Architecture and Code Optimization (TACO) ", December 2017, vol. 14, n^o 4, 27 p.

[25] I. AMER, C. LUCARZ, G. ROQUIER, M. MATTAVELLI, M. RAULET, J.-F. NEZAN, O. DEFORGES. *Reconfigurable video coding on multicore*, in "Signal Processing Magazine, IEEE", 2009, vol. 26, n^o 6, pp. 113–123, http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5230810

[26] S. ANANIAN. *The Static Single Information Form*, MIT, September 1999

[27] C. B. AOUN, L. ANDRADE, T. MAEHNE, F. PÊCHEUX, M.-M. LOUËRAT, A. VACHOUXY. *Pre-simulation elaboration of heterogeneous systems: The SystemC multi-disciplinary virtual prototyping approach*, in "Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS), 2015 International Conference on", IEEE, 2015, pp. 278–285

[28] D. BECKER, M. MOY, J. CORNET. *Parallel Simulation of Loosely Timed SystemC/TLM Programs: Challenges Raised by an Industrial Case Study*, in "MDPI Electronics", 2016, vol. 5, n^o 2, 22 p. [*DOI :* 10.3390/ELECTRONICS5020022], https://hal.archives-ouvertes.fr/hal-01321055

[29] N. BJØRNER, K. MCMILLAN, A. RYBALCHENKO. *On Solving Universally Quantified Horn Clauses*, in "Static Analysis: 20th International Symposium, SAS 2013, Seattle, WA, USA, June 20-22, 2013. Proceedings", Berlin, Heidelberg, F. LOGOZZO, M. FÄHNDRICH (editors), Springer Berlin Heidelberg, 2013, pp. 105–125, http://dx.doi.org/10.1007/978-3-642-38856-9_8

[30] D. CAROMEL, L. HENRIO. *A Theory of Distributed Objects*, Springer-Verlag, 2004

[31] P. COUSOT, R. COUSOT. *Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints*, in "4th ACM Symposium on Principles of Programming Languages (POPL'77)", Los Angeles, January 1977, pp. 238-252

[32] F. DE BOER, V. SERBANESCU, R. HÄHNLE, L. HENRIO, J. ROCHAS, C. C. DIN, E. BROCH JOHNSEN, M. SIRJANI, E. KHAMESPANAH, K. FERNANDEZ-REYES, A. M. YANG. *A Survey of Active Object Languages*, in "ACM Comput. Surv.", October 2017, vol. 50, n^o 5, pp. 76:1–76:39, http://doi.acm.org/10.1145/3122848

[33] M. DURANTON, D. BLACK-SCHAFFER, K. DE BOSSCHERE, J. MAEBE. *The HIPEAC VISION FOR ADVANCED COMPUTING IN HORIZON 2020, https://www.hipeac.net/v13*, 2013, https://www.hipeac.net/v13

[34] P. FEAUTRIER. *Scalable and Structured Scheduling*, in "International Journal of Parallel Programming", October 2006, vol. 34, n^o 5, pp. 459–487

[35] P. FEAUTRIER. *Dataflow analysis of array and scalar references*, in "International Journal of Parallel Programming", 1991, vol. 20, n^o 1, pp. 23–53

[36] P. FEAUTRIER, A. GAMATIÉ, L. GONNORD. *Enhancing the Compilation of Synchronous Dataflow Programs with a Combined Numerical-Boolean Abstraction*, in "CSI Journal of Computing", 2012, vol. 1, n^{o} 4, pp. 8:86–8:99, http://hal.inria.fr/hal-00860785

[37] K. FERNANDEZ-REYES, D. CLARKE, E. CASTEGREN, H.-P. VO. *Forward to a Promising Future*, in "Conference proceedings COORDINATION 2018", 2018

[38] R. FONTAINE, L. GONNORD, L. MOREL. *Polyhedral Dataflow Programming: a Case Study*, in "SBAC-PAD 2018 - 30th International Symposium on Computer Architecture and High-Performance Computing", Lyon, France, IEEE, September 2018, pp. 1-9, https://hal-cea.archives-ouvertes.fr/cea-01855997

[39] L. GONNORD, P. IANNETTA, L. MOREL. *Semantic Polyhedral Model for Arrays and Lists*, Inria Grenoble Rhône-Alpes ; UCBL ; LIP - ENS Lyon ; CEA List, June 2018, n^{o} RR-9183, https://hal.archives-ouvertes.fr/hal-01815759

[40] M. I. GORDON. *Compiler techniques for scalable performance of stream programs on multicore architectures*, Massachusetts Institute of Technology. Dept. of Electrical Engineering and Computer Science, 2010

[41] O. HAKJOO, L. WONCHAN, H. KIHONG, Y. HONGSEOK, Y. KWANGKEUN. *Selective context-sensitivity guided by impact pre-analysis*, in "ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI '14, Edinburgh, United Kingdom - June 09 - 11, 2014", ACM, 2014, 49 p.

[42] N. HALBWACHS, P. CASPI, P. RAYMOND, D. PILAUD. *The synchronous data flow programming language LUSTRE*, in "Proceedings of the IEEE", Sep 1991, vol. 79, n^{o} 9, pp. 1305-1320

[43] L. HENRIO. *Data-flow Explicit Futures*, I3S, Université Côte d'Azur, April 2018, https://hal.archives-ouvertes.fr/hal-01758734

[44] P. IANNETTA, L. GONNORD, L. MOREL, T. YUKI. *Semantic Array Dataflow Analysis*, Inria Grenoble Rhône-Alpes, December 2018, n^{o} RR-9232, pp. 1-22, https://hal.archives-ouvertes.fr/hal-01954396

[45] G. KAHN. *The semantics of a simple language for parallel programming*, in "Information processing", North-Holland, 1974

[46] M. MAALEJ, V. PAISANTE, P. RAMOS, L. GONNORD, F. PEREIRA. *Pointer Disambiguation via Strict Inequalities*, in "Code Generation and Optimisation", Austin, United States, February 2017, https://hal.archives-ouvertes.fr/hal-01387031

[47] M. MAALEJ KAMMOUN. *Low-cost memory analyses for efficient compilers*, Université Lyon 1, 2017, Thèse de doctorat, Université Lyon1, http://www.theses.fr/2017LYSE1167

[48] D. MONNIAUX, L. GONNORD. *Cell morphing: from array programs to array-free Horn clauses*, in "23rd Static Analysis Symposium (SAS 2016)", Edimbourg, United Kingdom, X. RIVAL (editor), Static Analysis Symposium, September 2016, https://hal.archives-ouvertes.fr/hal-01206882

[49] M. MOY. *Parallel Programming with SystemC for Loosely Timed Models: A Non-Intrusive Approach*, in "DATE", Grenoble, France, March 2013, 9 p. , https://hal.archives-ouvertes.fr/hal-00761047

[50] V. PAISANTE, M. MAALEJ, L. BARBOSA, L. GONNORD, F. M. Q. PEREIRA. *Symbolic Range Analysis of Pointers*, in "International Symposium of Code Generation and Optmization", Barcelon, Spain, March 2016, pp. 791-809, https://hal.inria.fr/hal-01228928

[51] A. PLESCO. *Program Transformations and Memory Architecture Optimizations for High-Level Synthesis of Hardware Accelerators*, Ecole normale supérieure de lyon - ENS LYON, September 2010, https://tel.archives-ouvertes.fr/tel-00544349

[52] P. QUINTON. *Automatic synthesis of systolic arrays from uniform recurrent equations*, in "ACM SIGARCH Computer Architecture News", 1984, vol. 12, n^{o} 3, pp. 208–214

[53] H. RIHANI, M. MOY, C. MAÏZA, R. I. DAVIS, S. ALTMEYER. *Response Time Analysis of Synchronous Data Flow Programs on a Many-Core Processor*, in "Proceedings of the 24th International Conference on Real-Time Networks and Systems", New York, NY, USA, RTNS '16, ACM, 2016, pp. 67–76, http://doi.acm.org/10.1145/2997465.2997472

[54] H. N. W. SANTOS, I. MAFFRA, L. OLIVEIRA, F. PEREIRA, L. GONNORD. *Validation of Memory Accesses Through Symbolic Analyses*, in "Proceedings of the 2014 ACM International Conference on Object Oriented Programming Systems Languages And Applications (OOPSLA'14)", Portland, Oregon, United States, October 2014, http://hal.inria.fr/hal-01006209

[55] W. THIES. *Language and compiler support for stream programs*, Massachusetts Institute of Technology, 2009

[56] A. TURJAN. *Compiling Nested Loop Programs to Process Networks*, Universiteit Leiden, 2007

[57] N. VENTROUX, T. SASSOLAS. *A new parallel SystemC kernel leveraging manycore architectures*, in "Design, Automation & Test in Europe Conference & Exhibition (DATE), 2016", IEEE, 2016, pp. 487–492

[58] S. VERDOOLAEGE. *Polyhedral Process Networks*, Handbook of Signal Processing Systems, Springer, 2010, pp. 931–965

[59] S. WILLIAMS, A. WATERMAN, D. PATTERSON. *Roofline: an insightful visual performance model for multicore architectures*, in "Communications of the ACM", 2009, vol. 52, n^{o} 4, pp. 65–76

[60] B. DA SILVA, A. BRAEKEN, E. H. D'HOLLANDER, A. TOUHAFI. *Performance modeling for FPGAs: extending the roofline model with high-level synthesis tools*, in "International Journal of Reconfigurable Computing", 2013, vol. 2013, 7 p.