RESEARCH CENTRE

Grenoble - Rhône-Alpes

IN PARTNERSHIP WITH:

CNRS, Ecole normale supérieure de Lyon, Université Claude Bernard (Lyon 1)

2020

ACTIVITY REPORT

Project-Team

CASH

Compilation and Analyses for Software and Hardware

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

DOMAIN

Algorithmics, Programming, Software and Architecture

THEME

Architecture, Languages and Compilation

Contents

Project-Team CASH

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

Keywords

Computer sciences and digital sciences

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. – Programming Languages

A2.1.1. – Semantics of programming languages

A2.1.2. – Imperative programming

A2.1.4. – Functional programming

A2.1.6. – Concurrent programming

A2.1.7. – Distributed programming

A2.1.10. – Domain-specific languages

A2.1.11. – Proof languages

A2.2. – Compilation

A2.2.1. – Static analysis

A2.2.2. – Memory models

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

A2.4.1. – Analysis

A2.4.3. – Proofs

A2.5.3. – Empirical Software Engineering

A2.5.4. – Software Maintenance & Evolution

A7.2.3. – Interactive Theorem Proving

Other research topics and application domains

B9.5.1. – Computer science

1 Team members, visitors, external collaborators

Research Scientists

- Christophe Alias [Inria, Researcher, HDR]
- Ludovic Henrio [CNRS, Researcher, HDR]
- Gabriel Radanne [Inria, from Dec 2020, Starting Faculty Position]
- Yannick Zakowski [Inria, Researcher, from Oct 2020]

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]
- Julien Emmanuel [Bull, CIFRE]
- Paul Iannetta [École Normale Supérieure de Lyon]
- Amaury Maille [École Normale Supérieure de Lyon]

Interns and Apprentices

- Baptiste Allorant [École Normale Supérieure de Lyon, from Apr 2020 until Jul 2020]
- Mickael Boichot [École Normale Supérieure de Lyon, from Jun 2020 until Aug 2020]
- Nicolas Chappe [École Normale Supérieure de Lyon, until Jun 2020]
- Avril De Goër De Herve [École Normale Supérieure de Lyon, from Apr 2020 until Jul 2020]
- Paul Geneau De Lamarliere [École Normale Supérieure de Lyon, from Apr 2020 until Jul 2020]
- Remy Neveu [École Normale Supérieure de Lyon, until Jun 2020]
- Hadrien Renaud [École polytechnique, from Apr 2020 until Jul 2020]

Administrative Assistants

- Solene Audoux [Inria, from Dec 2020]
- Sophie Gerard [Inria, until Nov 2020]

2 Overall objectives

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 [40] 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*).

2.1 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 [63] 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 [38] or Streamit [59], where the scheduling is fully static, and general communicating networks like KPNs [43] or RVC-Cal [21] 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 [28] that nowadays introduce more and more dataflow primitives to enable data-driven interactions between agents, particularly with *futures* [16, 32].

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

Short-term and ongoing activities. We obtained preliminary experimental [19, 18, 34] and theoretical [39] 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.

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.

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, 45], as well as tailoring static analyses for specialized applications in compilation [31, 57], Synchronous Dataflow scheduling [56], and extending the polyhedral model to irregular applications [17]. 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

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 [35]. 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.

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 [46], 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 [37] that tries to formalize the cost/precision compromise of interprocedural analyses with respect to a "context sensitivity parameter".

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 [22] 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 [27] 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 [61] and rely directly on array dataflow analysis [29]. 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 [30]. 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 [64] 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) [18]. 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 [36]. 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

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 [34]. 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.

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.

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) [18]. Hence, the very first step is then to chose an HLS tool and to investiguate 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 [64] 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 [54] 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

Short-term and ongoing activities. The HLS compiler designed in the CASH team currently extracts a fine-grain parallel intermediate representation (DPN [18, 19]) 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].

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 [20]. 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.

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 [58]. 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) [49] in SystemC [42] 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 [25]. 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

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 [62], 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.

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 [62] from CEA-LIST, some compute-intensive parts delegated to sc-during [48] from Matthieu Moy, and a co-simulation with external physical solvers using SystemC-MDVP [23] 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.

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 Application domains

The CASH team targets HPC programs, at different levels. Small computation kernels (tens of lines of code) that can be analyzed and optimized aggressively, medium-size kernels (hundreds of lines of code) that require modular analysis, and assembly of compute kernels (either as classical imperative programs or written directly in a dataflow language).

The work on various application domains and categories of programs is driven by the same idea: exploring various topics is a way to converge on unifying representations and algorithms even for specific applications. All these applications share the same research challenge: find a way to integrate computations, data, mapping, and scheduling in a common analysis and compilation framework.

Typical HPC kernels include linear solvers, stencils, matrix factorizations, BLAS kernels, etc. Many kernels can be found in the Polybench/C benchmark suite [52]. The irregular versions can be found in [53]. Numerical kernels used in quantitative finance [65] are also good candidates, e.g., finite difference and Monte-Carlo simulation.

The medium-size applications we target are streaming algorithms [21], scientific workflows [60], and also the now very rich domain of deep learning applications [44]. We explore the possibilities of writing (see Section 3.1) and compiling (see Section 3.3) applications using a dataflow language. As a first step, we will target dataflow programs written in SigmaC [24] for which the fine grain parallelism is not taken into account. In parallel, we will also study the problem of deriving relevant (with respect to safety or optimization) properties on dataflow programs with array iterators.

The approach of CASH is based on compilation, and our objective is to allow developers to design their own kernels, and benefit from good performance in terms of speed and energy efficiency without having to deal with fine-grained optimizations by hand. Consequently, our objective is first to improve the performance and energy consumption for HPC applications, while providing programming tools that can be used by developers and are at a convenient level of abstraction.

Obviously, large applications are not limited to assembly of compute kernels. Our languages and formalism definitions and analyses must also be able to deal with general programs. Our targets also include generalist programs with complex behaviors such as recursive programs operating on arrays, lists and trees; worklist algorithms (lists are not handled within the polyhedral domain). Analysis on these programs should be able to detect non licit memory accesses, memory consumption, hotspots, ..., and to prove functional properties.

The simulation activities are both applied internally in CASH, to simulate intermediate representations, and for embedded systems. We are interested in Transaction-Level Models (TLM) of Systems-on-a-Chip (SoCs) including processors and hardware accelerators. TLM provides an abstract but executable model of the chip, with enough details to run the embedded software. We are particularly interested in models written in a loosely timed coding style. We plan to extend these to heterogeneous simulations including a SystemC/TLM part to model the numerical part of the chip, and other simulators to model physical parts of the system.

5 Social and environmental responsibility

5.1 Footprint of research activities

Although we do not have a precise measure of our carbon (and other environmental) footprint, the two main sources of impact of computer-science research activities are usually transport (plane) and digital equipment (lifecycle of computers and other electronic devices).

Obviously, 2020 was a very particular year as we currently cannot do international travel. Many members of the CASH team are already in an approach of reducing their international travel, and hopefully the new solutions we had to set up to continue our activities during the COVID crisis will allow us to continue our research with a sustainable amount of travel, and using other forms of remote collaborations when possible.

As far as digital equipment is concerned, we try to extend the lifetime of our machines as much as possible.

5.2 Impact of research results

Many aspects of our research are meant to provide tools to make programs more efficient, in particular more power-efficient. It is very hard, however, to asses the actual impact of such research. In many cases, improvements in power-efficiency lead to a rebound effect which may weaken the benefit of the improvement, or even lead to an increase in total consumption (backfire).

CASH provides tools for developers, but does not develop end-user applications. We believe the social impact of our research depends more on the way developpers will use our tools than on the way we conduct our research. We do have a responsibility on the application domains we promote, though.

6 New software and platforms

6.1 New software

6.1.1 DCC

Name: DPN C Compiler

Keywords: Polyhedral compilation, Automatic parallelization, High-level synthesis

Functional Description: Dcc (Data-aware process network C Compiler) compiles a regular C kernel to a data-aware process network (DPN), a dataflow intermediate representation suitable for high-level synthesis in the context of high-performance computing. Dcc has been registered at the APP ("Agence de protection des programmes") and transferred to the XtremLogic start-up under an Inria license.

News of the Year: This year, Christophe implemented significant features in Dcc. First, channel systolization, a feature to reduce the overall size of the buffers. Second, a meta-compilation technique to speed-up the DPN parallelization. Finally, a dynamic deadlock detection feature to the thread model used for the debugging process.

Publication: hal-03143777

Authors: Christophe Alias, Alexandru Plesco

Contact: Christophe Alias

Participants: Christophe Alias, Alexandru Plesco

6.1.2 PoCo

Name: Polyhedral Compilation Library

Keywords: Polyhedral compilation, Automatic parallelization

Functional Description: PoCo (Polyhedral Compilation Library) is framework to develop program analysis and optimizations in the polyhedral model. PoCo features polyhedral building blocks as well as state-of-the-art polyhedral program analysis. PoCo has been registered at the APP (“agence de protection des programmes”) and transferred to the XtremLogic start-up under an Inria licence.

News of the Year: This year, Christophe implemented significant features in PoCo. First, an algorithm to derive an affine loop tiling from a regular kernel. Also, several polyhedral scheduling algorithms (greedy and shallow heuristics).

Author: Christophe Alias

Contact: Christophe Alias

Participant: Christophe Alias

6.1.3 MPPcodegen

Name: 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.

URL: `http://foobar.ens-lyon.fr/mppcodegen/`

Publication: hal-02493164

Authors: Christophe Alias, Guillaume Iooss, Sanjay Rajopadhye

Contacts: Christophe Alias, Guillaume Iooss, Sanjay Rajopadhye

Partner: Colorado State University

6.1.4 Encore with dataflow explicit futures

Keywords: Language, Optimizing compiler, Source-to-source compiler, Compilers

Functional Description: Fork of the Encore language compiler, with a new "Flow" construct implementing data-flow explicit futures.

URL: `https://gitlab.inria.fr/lhenrio/encorewithdatafuts`

Contacts: Ludovic Henrio, Matthieu Moy

6.1.5 odoc

Keyword: Ocaml

Functional Description: OCaml is a statically typed programming language with wide-spread use in both academia and industry. Odoc is a tool to generate documentation of OCaml libraries, either as HTML websites for online distribution or to create PDF manuals and man pages.

News of the Year: This year, Gabriel Radanne rewrote a significant portion of odoc to provide improved HTML output, make it possible to produce other document formats, and introduce the ability to produce man pages. Florian Angeletti then implemented PDF output and integrated the usage of odoc in the official OCaml distribution. Concurrently, Jon Ludlam and Leo White rewrote the resolution mechanism of odoc. This all led to a joint presentation at the OCaml workshop and an upcoming new major release.

URL: `https://github.com/ocaml/odoc/`

Contacts: Jon Ludlam, Gabriel Radanne

Participants: Jon Ludlam, Gabriel Radanne, Florian Angeletti, Leo White

6.1.6 calv

Name: AVL calculator

Keywords: Data structures, OpenMP

Functional Description: calv is a calculator which is used to run different implementations of AVL trees, and compare their relative performances.

URL: `https://gitlab.inria.fr/paiannet/calv`

Contact: Paul Iannetta

6.1.7 mppcheck

Keywords: Polyhedral compilation, Program verification

Functional Description: mppcheck features a pragma language to specify a polyhedral program transformation directly in the code and a verification algorithm able to check the correctness of the specified transformation. Our language is general enough to specify a loop tiling by an arbitrary polyhedral tile shape (e.g., hexagons, diamonds, trapezoids), and whose size may depend on a scaling parameter (monoparametric tiling). Our verification algorithm checks the legality of the proposed transformation, and provides counterexamples of unsatisfied dependences when it is incorrect. In addition, out tool infers the domain of scaling parameters where the tiling is not legal.

URL: `http://foobar.ens-lyon.fr/mppcheck/`

Publication: hal-03106070

Authors: Christophe Alias, Guillaume Iooss, Sanjay Rajopadhye

Contacts: Christophe Alias, Guillaume Iooss, Sanjay Rajopadhye

Participants: Christophe Alias, Guillaume Iooss, Sanjay Rajopadhye

Partner: Colorado State University

6.1.8 fkcc

Name: The Farkas Calculator

Keywords: DSL, Farkas Lemma, Polyhedral compilation

Scientific Description: fkcc is a scripting tool 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.

Functional Description: fkcc is a scripting tool 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.

Release Contributions: - Script language - Polyhedral constructors - Farkas summation solver

News of the Year: This year, Christophe improved the projection algorithm invoked by the 'keep' keyword by combining a Gaussian elimination with a Fourier-Motzkin elimination. Compared to Chernikova algorithm, the performances are greatly improved.

URL: `http://foobar.ens-lyon.fr/fkcc/`

Publication: hal-03106000

Author: Christophe Alias

Contact: Christophe Alias

Participant: Christophe Alias

6.1.9 Vellvm

Keywords: Coq, Semantic, Compilation, Proof assistant, Proof

Scientific Description: A modern formalization in the Coq proof assistant of the sequential fragment of LLVM IR. The semantics, based on the Interaction Trees library, presents several rare properties for mechanized development of this scale: it is compositional, modular, and extracts to a certified executable interpreter. A rich equational theory of the language is provided, and several verified tools based on this semantics are in development.

Functional Description: Formalization in the Coq proof assistant of a subset of the LLVM compilation infrastructure.

URL: `https://github.com/vellvm/vellvm`

Contacts: Yannick Zakowski, Steve Zdancewic

Participants: Yannick Zakowski, Steve Zdancewic, Calvin Beck, Irene Yoon

Partner: University of Pennsylvania

7 New results

This section presents the scientific results obtained in the evaluation period. They are grouped according to the directions of our research program.

7.1 Direction 1: Definition of Dataflow Representations of Parallel Programs

7.1.1 Dataflow-explicit futures

Participants Ludovic Henrio, Matthieu Moy, Amaury Maillé, Nicolas Chappe.

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 [39, 33], 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.

In 2020, we continued our work on the implementation of data-flow explicit futures in the Encore compiler. We also showed that the `forward` [32] construct proposed in the context of control-flow futures was semantically equivalent to the usual `return` construct in the context of data-flow explicit futures. We submitted this work to the <Programming> journal, and submitted a major revision, currently under review.

7.1.2 Promise Plus: Flexible Synchronization for Parallel Computations on Arrays

Participants Ludovic Henrio, Matthieu Moy, Amaury Maillé.

Parallel applications make use of parallelism where work is shared between tasks; often, tasks need to exchange data stored in arrays and synchronize depending on the availability of these data. Fine-grained synchronizations, *e.g.* one synchronization for each element in the array, may lead to too many synchronizations while coarse-grained synchronizations, *e.g.* a single synchronization for the whole array, may prevent parallelism. We propose PromisePlus, a synchronization tool allowing tasks to synchronize on chunks of arrays with a granularity configurable by the programmer.

Results are still preliminary, but promising, and have been accepted for publication at the FSEN 2021 conference [11]. We are currently working on an extension where the granularity of synchronization can vary at runtime.

7.1.3 Distributed futures

Participants 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 is realised in collaboration with Pierre Leca and Wijnand Suijlen (Huawei Technologies), and Françoise Baude (Université Côte d'Azur, CNRS, I3S). Pierre Leca defended his PhD Thesis in October 2020.

7.1.4 Locally abstract globally concrete semantics

Participants 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, this work has slowed down in 2020, partly because the visit (as invited professor) od Reiner Hahnle had to be postponned. The final version of the article is almost ready for a first submission.

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).

7.1.5 PNets: Parametrized networks of automata

Participants 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 polished during the year and submitted to LMCS in September 2020.

These work should be continued in 2021 and an internship student will be co-advised by ric Madelaine, Rabéa Ameur Boulifa, and Ludovic Henrio. The subject of the internship is on "refinement for open automata".

7.1.6 A Survey on Verified Reconfiguration

Participants 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 2021.

7.1.7 A Survey on Parallelism and Determinacy

Participants Ludovic Henrio, Laure Gonnord, Christophe Alias, Gabriel Radanne.

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 survey is in the last phase of writing before submission in Spring 2021

This work, coordinated by Laure Gonnord and Ludovic Henrio, also involves contributors outside the CASH team. For the moment Gabriel Radanne (Initially Inria Paris, but joined CASH in 2020) and Lionel Morel (CEA).

7.2 Direction 2: Expressivity and Scalability of Static Analyses

7.2.1 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 [26] by using the ideas from the cell abstraction method of [47] 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 short versionof the technique has been accepted in [5] The full contribution will be submitted soon.

7.2.2 Pass Neutralization in LLVM

Participants Laure Gonnord, Avril De Goer de Hervé.

In the context of the CAPESA project, Sebastien Mosser, Jean Privat and Laure Gonnordcoadvised the master internship of Avril De Goer de Hervé. The objective was to work on the definition and implementation of Pass Neutralization inside the LLVM compiler.

We proposed a dynamic approach for measuring impact of passes improvement in theLLVMcompiler. Pursuing the previous work on LLVM cartography, we propose a refinement of the previous collectedi nformation with fine-grain dynamic analyses by pass neutralization, for which we propose a general methodology.

A technical report has been produced. We submitted an extended version to the SLE conference, but unfortunately the paper was rejected. We plan to do further experiments before re-submitting.

7.2.3 An incremental type-checker for OCaml modules

Participants Gabriel Radanne, Didier Remy, Jacques Garrigue *(University of Nagoya)*.

Modules are a core feature of ML languages, allowing to assemble pieces of software in a high-level and composable fashion. OCaml benefits from a particularly rich module system which was originally described more than two decades ago, but has significantly grown since.

This year, Gabriel Radanne, in collaboration with Didier Rémy and Jacques Garrigue, started formalizing a new module system which combines all of the features that have been introduced since the last formalization effort by Xavier Leroy. This new system also improves inference and provides a solid basis for further experiments, such as the “modular implicits” that are currently being investigated in the cambium inria team. Gabriel Radanne started a “clean room” implementation of a prototype type-checker for this new module system.

7.2.4 A formal, compositional, modular and executable semantics for LLVM IR

Participants Yannick Zakowski, Calvin Beck *(University of Pennsylvania)*, Irene Yoon *(University of Pennsylvania)*, Steve Zdancewic *(University of Pennsylvania)*.

The industrial-strength compilation infrastructure LLVM is particularly centered around an informally defined intermediate representation named LLVM IR. The correctness of all optimizations, static analyzes and tools used at the core of LLVM hence rely on a fine understanding of the semantics of this SSA-based language.

Yannick Zakowski, in collaboration with Calvin Beck, Irene Yoon and Steve Zdancewic from the University of UPenn, has developed in the Coq proof assistant a new formal semantics for a large subset of sequential LLVM IR. In contrast with previous formal works, this semantics brings modern semantic approaches – denotational and exploiting algebraic-effects – to the formal verification of realistic languages. This feat is achieved by building on the Interaction Trees library that Yannick Zakowski has developed with co-authors a year prior.

This work is currently submitted at PLDI'21. The associated artifact is available publicly: `https://github.com/vellvm/vellvm/tree/artifact`.

7.2.5 An equational proof of correctness for the HELIX backend

Participants Yannick Zakowski, Calvin Beck *(University of Pennsylvania)*, Irene Yoon *(University of Pennsylvania)*, Ilia Zaichuk *(Di Gamma)*, Vadim Zaliva *(Carneggie Mellon University)*, Steve Zdancewic *(University of Pennsylvania)*.

The SPIRAL project, developped over the last twenty years, is a compilation infrastructure to synthesize high performance code for numerical computations from high level mathematical specifications. The HELIX project is a recent effort to formalize in the Coq proof assistant part of this infrastructure, developped by Vadim Zaliva during his PhD.

Yannick Zakowski has written with his collaborators a backend for HELIX by compiling down to Vellvm, the formalization of LLVM IR he has developed. They proved most of this compiler correct – some operators are not yet covered. The same way Vellvm introduces modern semantic techniques, this proof of correctness introduces modern proof techniques: we reason compositionaly and equationaly using a termination sensitive relational program logic, in stark contrast with the traditional simulation diagrams.

This work is currently submitted at ITP'21. The associated artifact is available publicly: `https://github.com/vzaliva/helix/tree/itp21`.

7.3 Direction 3: Compiling and Scheduling Dataflow Programs

7.3.1 FKCC : the Farkas Calculator

Participants 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 IMPACT'20 workshop [2].

7.3.2 On the Verification of Polyhedral Program Transformations

Participants Christophe Alias, Guillaume Iooss, Sanjay Rajopadhye.

This work presents a pragma language to specify a polyhedral program transformation directly in the code and a verification algorithm able to check the correctness of the specified transformation. Our language is general enough to specify a loop tiling by an arbitrary polyhedral tile shape (e.g., hexagons, diamonds, trapezoids), and whose size may depend on a scaling parameter (monoparametric tiling). Our verification algorithm checks the legality of the proposed transformation, and provides counterexamples of unsatisfied dependences when it is incorrect. In addition, out tool infers the domain of scaling parameters where the tiling is not legal. We developed a tool suite implementing these concepts with a verification tool (mppcheck) and a code generation tool (mppcodegen), that are available and may be downloaded together with a rich set of examples. We evaluate the performance of the verification and the code generation on kernels from the PolyBench suite.

This work is achieved in collaboration with Guillaume Iooss (CORSE team) and Sanjay Rajopadhye (Colorado State University) and will be presented as a regular paper in the HPCS'20 conference (special session CADO) [3].

7.3.3 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 results of the experimentation and the new data layout have been accepted at the COMPAS french conference[12] and as a short CGO paper. Further promising experiments with openMP and a benchmark coming from the ocaml typer (with Gabriel Radanne) are currently done.

This work is done in collaboration with Lionel Morel (CEA Grenoble), coadvisor of the PhD of Paul Iannetta.

7.3.4 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 [41]. 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. Code sheduling is under progress.

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

7.4 Direction 4: HLS-specific Dataflow Optimizations

7.4.1 Data-aware Process Networks

Participants Christophe Alias, Alexandru Plesco.

With the emergence of reconfigurable FPGA circuits as a credible alternative to GPUs for HPC acceleration, new compilation paradigms are required to map high-level algorithmic descriptions to a circuit configuration (High-Level Synthesis, HLS). In particular, novel parallelization algorithms and intermediate representations are required. In this work, we present the data-aware process networks (DPN), a dataflow intermediate representation suitable for HLS in the context of high-performance computing. DPN combines the benefits of a low-level dataflow representation – close to the final circuit – and affine iteration space tiling to explore the parallelization trade-offs (local memory size, communication volume, parallelization degree). We outline our compilation algorithms to map a C program to a DPN (front-end), then to map a DPN to an FPGA configuration (back-end). Finally, we present synthesis results on compute-intensive kernels from the Polybench suite.

The results of this work have been transferred to the XtremLogic startup, it will be presented as a regular paper in the CC'21 conference.

7.5 Direction 5: Simulation of Hardware

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

Participants 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). In 2020, we extended the approach to allow running full-stack simulation including hardware, advanced operating system such as Linux, and application, which raises new issues (virtual memory, heavy use of lockless synchronization in the kernel, ...).

This work was be published at the ASP-DAC 2020 conference [6], and an extended journal version was accepted in 2021. The Ph.D Thesis of Gabriel Busnot on the subject was defended in December 2020.

7.5.2 Simulation of the Portals 4 protocol, and case study on the BXI interconnect

Participants Julien Emmanuel, Ludovic Henrio, Matthieu Moy.

In the context of Julien Emmanuel's CIFRE Ph.D, in collaboration with Atos/Bull, we present a new network simulator, which models the Portals 4 communication protocol used in High Performance Computing (HPC). It is built on top of SimGrid and uses cooperative actors to model the interactions between compute nodes in a supercomputer. Unlike most simulators in HPC, it models both communications on the interconnect and on the PCIe network inside each compute node, whithout going for a full emulation of the hardware. The simulator can be used to optimize or debug an application without having to use an actual supercomputer. This is made possible by leveraging SimGrid's flow model and it enables accurate simulation with good performances, even when running the model on a laptop. We test this simulator with custom experiments as well as existing Portals code, and compare the results with Portals executions on an actual cluster using Atos' BXI interconnect. The simulator is currently being extended to support unmodified MPI applications.

This work was published at the MSPDS workshop of the HPCS conference [8].

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

Participants Matthieu Moy.

In RTNS 2016, Rihani et al. [55] proposed an algorithm to compute the impact of interference on memory accesses on the timing of a task graph. It calculates a static, time-triggered schedule, i.e. a release date and a worst-case response time for each task. The task graph is a DAG, typically obtained by compilation of a high-level dataflow language, and the tool assumes a previously determined mapping and execution order. The algorithm is precise, but suffers from a high $O(n^4)$ complexity, n being the number of input tasks. Since we target many-core platforms with tens or hundreds of cores, applications likely to exploit the parallelism of these platforms are too large to be handled by this algorithm in reasonable time.

This paper proposes a new algorithm that solves the same problem. Instead of performing global fixed-point iterations on the task graph, we compute the static schedule incrementally, reducing the complexity to $O(N^2)$. Experimental results show a reduction from 535 seconds to 0.90 seconds on a benchmark with 384 tasks, i.e. 593 times faster.

This work was published at DATE 2020 [7].

8 Bilateral contracts and grants with industry

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

9 Partnerships and cooperations

9.1 International initiatives

CAPESA

Title: *CharActerisation of Program Evolution with Static Analyses*

Duration: 2020 - *2022*

Coordinator: Laure Gonnord

Partners:

- Computer Science Department at Université du Québec à Montréal (UQAM), ACE research group (https://ace-design.github.io/)., Université du Québec À Montréal (Canada)

Inria contact: Laure Gonnord

Summary: In this project we propose to study code transformation in terms of "semantic diff". This notion will be defined thanks to code intermediate representations such as Abstract Syntax Trees (AST) or the control flow graph, not by textual representation. The objective is not only to compute but also to manipulate these "diffs" in several contexts: being able to reapply a diff on another program than the one it comes from, quantify the interference between two diffs, and more generally to study the composability of several diffs. The approach will be experimentally validated on problems coming from the domain of expertise of both teams of the cooperation: compiler pass analyses (expertise of CASH), and git commits (expertise of UQAM). The complementarity of the analysis and compilation approaches of the CASH team and the expertise on software engineering of the UQAM member will ensure the success and the originality of the project.

9.1.1 Participation in other international programs

PolyTrace exploratory action

Title: PolyTrace – Compiling from execution traces

Duration: 2020 - *2024*

Coordinator: Christophe Alias

Partners:

- Prof. Keiji Kimura, Waseda University, Tokyo, Japan

Inria contact: Christophe Alias

Summary: In this project, we propose a new paradigm for compiler optimization: given several optimized execution traces, *learn* a compiler optimization which produces the same effects (same schedule, same parallelism, same memory allocation). We focus on regular programs and polyhedral compilation, where we believe the properties of regularity enable such an inference. The approach will be applied to buffer sizing, then to polyhedral scheduling.

9.1.2 ANR

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

10 Dissemination

10.1 Promoting scientific activities

10.1.1 Scientific events: organisation

General chair, scientific chair

- Laure Gonnord is in the steering commitee of the NSAD (Numerical and Symbolic Abstract Domains), from 2019

10.1.2 Scientific events: selection

Chair of conference program committees

- Christophe Alias was PC chair for the computer architecture track of Compas'20, a French conference on computer architecture and operating systems.

Member of the conference program committees

- Matthieu Moy was PC member at Emsoft'2020
- Laure Gonnord was PC member at POPL'2021 and CC'2021

10.1.3 Invited talks

- Laure Gonnord was invited in the IRIF "PPS Seminar" in November 2020.

10.1.4 Leadership within the scientific community

- Laure Gonnord belongs to the national board of the GDR GPL (starting 2021) and chair of the associated Junior School.
- Ludovic Henrio is now leading the new compilation and language group of the GDR GPL (starting 2021).

10.1.5 Scientific expertise

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

10.2 Teaching - Supervision - Juries

10.2.1 Teaching

- Bachelor ("Licence"):
  - Christophe Alias, Compilation, CM+TD, 27h, 3A, INSA Centre Val de Loire. (2020)
  - Matthieu Moy, Concurrent Programming, CM+TD+TP, 57h, L3, UCBL. (2020)
  - Matthieu Moy, Recursive Programming, TD+TP, 28h, L1, UCBL. (2020)
  - Matthieu Moy, Git, CM+TP: 12h, L3, UCBL. (2020)
  - Amaury Maillé, Concurrent Programming, 8h TD, 16h TP, L3, UCBL. (2020)
  - Amaury Maillé, Algorithms and Object-Oriented Programming, 28h TP, L3, UCBL. (2020)
  - Julien Emmanuel, Programmation fonctionnelle pour le WEB, TP, 28h, L2, UCBL. (2020)

- Julien Emmanuel, Base de données et Programmation WEB, TP, 22h, L2, UCBL. (2020)
- Julien Braine, Architecture et Systèmes (ASR1), TP/TD 32h. L3, ENSL (2020)
- Laure Gonnord, Systèmes (ASR 5), CM+TD+TP (half online), 29h, L2, Lyon1

• Master:

- Christophe Alias, Compiler optimizations for embedded applications, CM+TD, 27h, 4A, INSA Centre Val de Loire. (2020)
- Christophe Alias and Matthieu Moy, Hardware Compilation and Simulation, CM+TD, 32h, M2 Informatique Fondamentale, ENS de Lyon. (2020)
- Matthieu Moy, Software Engineering, CM+TD+TP, 25h, M1, UCBL. (2020)
- Paul Iannetta, Software Engineering, TP, 16.5h, M1, UCBL. (2020)
- Matthieu Moy, Compilation and program transformations, TD+TP, 22,5h, CM: 7.5h. M1, UCBL. (2020, Sept-Jan 21)
- Matthieu Moy, Compilation and program transformations, TD+TP, 22,5h, CM: 7.5h. M1, UCBL. (2020, Jan: intensive course)
- Laure Gonnord, Préparation au concours du CAPES-NSI, 50h, CM, TD, TP, Lyon1 (2020).
- Laure Gonnord, Ludovic Henrio, Gabriel Radanne and Yannick Zakowski, Compilation and Program Analysis, video recording and online courses, total 26 hours CM, ENSL (2020)
- Paul Iannetta and Gabriel Radanne, Compilation and Program Analysis, online labs, 28h each (2020).
- Amaury Maillé, Advanced Programming, 15 h TP, M1, UCBL. (2020)
- Ludovic Henrio, "An algorithmic approach to distributed systems", 6h (CM+TD), M2, University of Nice Sophia-Antipolis.

10.2.2 Supervision

• Defended PhD: 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 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).

• Defended PhD: 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).

10.2.3 Juries

• Christophe Alias was reviewer for the PhD thesis of Harenome Razanajato, "Polyhedral Code Generation: Reducing Overhead and Increasing Parallelism", University of Strasbourg (September 2020)

• Christophe Alias is oral examiner for the second concours of ENS de Lyon.

- Laure Gonnord was jury member for the Concours d'admission de l'Agrégation de Sciences Industrielles, spécialité Informatique Industrielle, in June 2019 and 2020.
- Laure Gonnord was jury member for the Junior Inria research positions selection of the Bordeaux Inria center and the National Inria selection in Spring 2020.
- Laure Gonnord was external jury member for the PhD defense of Nicolas Szlifierski, IMT Atlantique. "Contrôle sûr de chaînes d'obfuscations logicielles" (December 2020)
- Laure Gonnord was external jury member for the PhD defense of Rémy Grüblatt, Univ. Lyon1. "From WiFi Performance Evaluation to Controlled Mobility in Drone Networks" (January 2021)
- Matthieu Moy was reviewer for the Ph.D of Riyane SID LAKHDAR, "Methodology for Code-Optimization of Memory Data-Layouts for High-Performance-System Architectures with Complex Memory Hierarchies" (CEA Grenoble).

10.3 Popularization

10.3.1 Education

- Video "Polyèdres et synthèse de programmes" by Christophe Alias for the cycle de conférences NSI, sponsored by Inria for the purpose of high-school teachers.

11 Scientific production

11.1 Publications of the year

International journals

[1] L. Henrio, C. Kessler and L. Li. 'Leveraging access mode declarations in a model for memory consistency in heterogeneous systems'. In: *Journal of Logical and Algebraic Methods in Programming* 110 (Jan. 2020), pp. 1–17. DOI: 10.1016/j.jlamp.2019.100498. URL: https://hal.archives-ouvertes.fr/hal-02331964.

International peer-reviewed conferences

[2] C. Alias. 'Farkas Lemma made easy'. In: 10th International Workshop on Polyhedral Compilation Techniques (IMPACT 2020). Bologna, Italy, 22nd Dec. 2019, pp. 1–6. URL: https://hal.inria.fr/hal-02422033.

[3] C. Alias, G. Iooss and S. Rajopadhye. 'On the Verification of Polyhedral Program Transformations'. In: 18th International Conference on High Performance Computing & Simulation (HPCS 2020), 3rd Special Session on Compiler Architecture, Design and Optimization (CADO 2020). Barcelona, Spain, 25th Jan. 2020. URL: https://hal.archives-ouvertes.fr/hal-03106070.

[4] C. Alias and A. Plesco. 'Data-Aware Process Networks'. In: Proceedings of the 30th ACM SIGPLAN International Conference on Compiler Construction. Virtual, South Korea, 2nd Mar. 2021. DOI: 10.1145/3446804.3446847. URL: https://hal.inria.fr/hal-03143777.

[5] J. Braine and L. Gonnord. 'Proving array properties using data abstraction'. In: Numerical and Symbolic Abstract Domains (NSAD). Virtual, United States, 24th Sept. 2020. URL: https://hal.archives-ouvertes.fr/hal-02948081.

[6] G. Busnot, T. Sassolas, N. Ventroux and M. Moy. 'Standard-compliant Parallel SystemC simulation of Loosely-Timed Transaction Level Models'. In: ASP-DAC 2020 - 25th Asia and South Pacific Design Automation Conference. Beijing, China: https://aspdac2020.github.io/aspdac20/, 13th Jan. 2020, pp. 1–6. URL: https://hal.archives-ouvertes.fr/hal-02416253.

[7] M. Dupont De Dinechin, M. Schuh, M. Moy and C. Maïza. 'Scaling Up the Memory Interference Analysis for Hard Real-Time Many-Core Systems'. In: DATE 2020 - Design, Automation and Test in Europe Conference. Grenoble, France, 9th Mar. 2020, pp. 1–4. URL: https://hal.archives-ouvertes.fr/hal-02431273.

[8] J. Emmanuel, M. Moy, L. Henrio and G. Pichon. 'Simulation of the Portals 4 protocol, and case study on the BXI interconnect'. In: HPCS 2020 - International Conference on High Performance Computing & Simulation. Barcelona, Spain: http://hpcs2020.cisedu.info, 10th Dec. 2020, pp. 1–8. URL: https://hal.archives-ouvertes.fr/hal-02972297.

[9] L. Henrio, E. B. Johnsen and V. K. I. Pun. 'Active Objects with Deterministic Behaviour'. In: *Integrated Formal Methods. IFM 2020.* Integrated Formal Methods. IFM 2020. Lugano, Switzerland, 13th Nov. 2020, pp. 181–198. DOI: 10.1007/978-3-030-63461-2_10. URL: https://hal.archives-ouvertes.fr/hal-03008405.

[10] P. Leca, W. Suijlen, L. Henrio and F. Baude. 'Distributed futures for efficient data transfer between parallel processes'. In: SAC 2020 - 35th ACM/SIGAPP Symposium On Applied Computing. Brno, Czech Republic, 30th Mar. 2020. DOI: 10.1145/3341105.3374104. URL: https://hal.archives-ouvertes.fr/hal-02417953.

[11] A. Maillé, L. Henrio and M. Moy. 'Promise Plus: Flexible Synchronization for Parallel Computations on Arrays'. In: *Lecture Notes in Computer Science.* 9th IPM International Conference on Fundamentals of Software Engineering. Tehran, Iran, 19th May 2021. URL: https://hal.archives-ouvertes.fr/hal-03143269.

Conferences without proceedings

[12] P. Iannetta, L. Gonnord and L. Morel. 'On optimizing scalar self-rebalancing trees'. In: COMPAS 2020 - Conférence francophone d'informatique en Parallélisme, Architecture et Système. Lyon, France, 2020. URL: https://hal.archives-ouvertes.fr/hal-03048742.

Reports & preprints

[13] P. Iannetta, L. Gonnord and L. Morel. *On optimizing scalar self-rebalancing trees.* LYON, France: INRIA , LIP; Inria - Research Centre Grenoble – Rhône-Alpes; Université de Lyon I Claude Bernard, 2020. URL: https://hal.inria.fr/hal-02573052.

[14] G. Iooss, C. Alias and S. Rajopadhye. *Monoparametric Tiling of Polyhedral Programs.* 3rd Jan. 2021. URL: https://hal.inria.fr/hal-02493164.

11.2 Other

Softwares

[15] [SW] H. Renaud, *Fut on Flow*, 6th Aug. 2020. HAL: ⟨hal-02908763⟩, URL: https://hal.archives-ouvertes.fr/hal-02908763, VCS: https://gitlab.inria.fr/datafut/fut-on-flow, SWHID: ⟨swh:1:dir:409184eb3c27c62e3ff996e1b1a7baf41c5b3153;origin=https://hal.archives-ouvertes.fr/hal-02908763;visit=swh:1:snp:9a972df7a9a7d45584256fc0722e03a27d211154;anchor=swh:1:rev:fc074c0539a309fefc4d51c77f5816bc054a5fd6;path=/⟩.

11.3 Cited publications

[16] D. Caromel and L. Henrio. *A Theory of Distributed Objects.* Springer-Verlag, 2004.

[17] C. Alias, A. Darte, P. Feautrier and L. Gonnord. 'Multi-dimensional Rankings, Program Termination, and Complexity Bounds of Flowchart Programs'. In: *International Static Analysis Symposium (SAS'10).* 2010.

[18] C. Alias and A. Plesco. *Data-aware Process Networks.* Research Report RR-8735. Inria - Research Centre Grenoble – Rhône-Alpes, June 2015, p. 32. URL: https://hal.inria.fr/hal-01158726.

[19] C. Alias and A. Plesco. *Method of Automatic Synthesis of Circuits, Device and Computer Program associated therewith*. Patent FR1453308. Apr. 2014.

[20] C. Alias and A. Plesco. 'Optimizing Affine Control with Semantic Factorizations'. In: *ACM Transactions on Architecture and Code Optimization (TACO)* 14.4 (Dec. 2017), p. 27.

[21] I. Amer, C. Lucarz, G. Roquier, M. Mattavelli, M. Raulet, J.-F. Nezan and O. Deforges. 'Reconfigurable video coding on multicore'. In: *Signal Processing Magazine, IEEE* 26.6 (2009), pp. 113–123. URL: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5230810.

[22] S. Ananian. 'The Static Single Information Form'. MA thesis. MIT, Sept. 1999.

[23] C. B. Aoun, L. Andrade, T. Maehne, F. Pêcheux, M.-M. Louërat and 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.

[24] P. Aubry, P.-E. Beaucamps, F. Blanc, B. Bodin, S. Carpov, L. Cudennec, V. David, P. Doré, P. Dubrulle, B. Dupont De Dinechin, F. Galea, T. Goubier, M. Harrand, S. Jones, J.-D. Lesage, S. Louise, N. Morey Chaisemartin, T. H. Nguyen, X. Raynaud and R. Sirdey. 'Extended Cyclostatic Dataflow Program Compilation and Execution for an Integrated Manycore Processor'. In: *Alchemy 2013 - Architecture, Languages, Compilation and Hardware support for Emerging ManYcore systems*. Vol. 18. Proceedings of the International Conference on Computational Science, ICCS 2013. Barcelona, Spain, June 2013, pp. 1624–1633. DOI: 10.1016/j.procs.2013.05.330. URL: https://hal.inria.fr/hal-00832504.

[25] D. Becker, M. Moy and J. Cornet. 'Parallel Simulation of Loosely Timed SystemC/TLM Programs: Challenges Raised by an Industrial Case Study'. In: *MDPI Electronics* 5.2 (2016). Ed. by F. Rousseau, G. Nicolescu, A. Baghdadi and M. Bassiouni, p. 22. DOI: 10.3390/electronics5020022. URL: https://hal.archives-ouvertes.fr/hal-01321055.

[26] N. Bjørner, K. McMillan and A. Rybalchenko. 'On Solving Universally Quantified Horn Clauses'. In: *Static Analysis: 20th International Symposium, SAS 2013, Seattle, WA, USA, June 20-22, 2013. Proceedings*. Ed. by F. Logozzo and M. Fähndrich. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 105–125. URL: http://dx.doi.org/10.1007/978-3-642-38856-9_8.

[27] P. Cousot and 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, Jan. 1977, pp. 238–252.

[28] F. De Boer, V. Serbanescu, R. Hähnle, L. Henrio, J. Rochas, C. C. Din, E. Broch Johnsen, M. Sirjani, E. Khamespanah, K. Fernandez-Reyes and A. M. Yang. 'A Survey of Active Object Languages'. In: *ACM Comput. Surv.* 50.5 (Oct. 2017), 76:1–76:39. DOI: 10.1145/3122848. URL: http://doi.acm.org/10.1145/3122848.

[29] P. Feautrier. 'Dataflow analysis of array and scalar references'. In: *International Journal of Parallel Programming* 20.1 (1991), pp. 23–53.

[30] P. Feautrier. 'Scalable and Structured Scheduling'. In: *International Journal of Parallel Programming* 34.5 (Oct. 2006), pp. 459–487.

[31] P. Feautrier, A. Gamatié and L. Gonnord. 'Enhancing the Compilation of Synchronous Dataflow Programs with a Combined Numerical-Boolean Abstraction'. In: *CSI Journal of Computing* 1.4 (2012), 8:86–8:99. URL: http://hal.inria.fr/hal-00860785.

[32] K. Fernandez-Reyes, D. Clarke, E. Castegren and H.-P. Vo. 'Forward to a Promising Future'. In: *Conference proceedings COORDINATION 2018*. 2018.

[33] K. Fernandez-Reyes, D. Clarke, L. Henrio, E. Broch Johnsen and T. Wrigstad. 'Godot: All the Benefits of Implicit and Explicit Futures'. In: *ECOOP 2019 - 33rd European Conference on Object-Oriented Programming*. Leibniz International Proceedings in Informatics (LIPIcs). London, United Kingdom, July 2019, pp. 1–28. URL: https://hal.archives-ouvertes.fr/hal-02302214.

[34] R. Fontaine, L. Gonnord and 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, Sept. 2018, pp. 1–9. URL: https://hal-cea.archives-ouvertes.fr/cea-01855997.

[35] L. Gonnord, P. Iannetta and L. Morel. *Semantic Polyhedral Model for Arrays and Lists*. Research Report RR-9183. Inria Grenoble Rhône-Alpes ; UCBL ; LIP - ENS Lyon ; CEA List, June 2018. URL: https://hal.archives-ouvertes.fr/hal-01815759.

[36] M. I. Gordon. ‘Compiler techniques for scalable performance of stream programs on multicore architectures’. PhD thesis. Massachusetts Institute of Technology. Dept. of Electrical Engineering and Computer Science, 2010.

[37] O. Hakjoo, L. Wonchan, H. Kihong, Y. Hongseok and 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, p. 49.

[38] N. Halbwachs, P. Caspi, P. Raymond and D. Pilaud. ‘The synchronous data flow programming language LUSTRE’. In: *Proceedings of the IEEE* 79.9 (Sept. 1991), pp. 1305–1320.

[39] L. Henrio. *Data-flow Explicit Futures*. Research Report. I3S, Université Côte d’Azur, Apr. 2018. URL: https://hal.archives-ouvertes.fr/hal-01758734.

[40] M. Duranton, D. Black-Schaffer, K. De Bosschere and J. Maebe. *The HIPEAC VISION FOR ADVANCED COMPUTING IN HORIZON 2020, https://www.hipeac.net/v13*. 2013. URL: https://www.hipeac.net/v13.

[41] P. Iannetta, L. Gonnord, L. Morel and T. Yuki. *Semantic Array Dataflow Analysis*. Research Report RR-9232. Inria Grenoble Rhône-Alpes, Dec. 2018, pp. 1–22. URL: https://hal.archives-ouvertes.fr/hal-01954396.

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

[43] G. Kahn. ‘The semantics of a simple language for parallel programming’. In: *Information processing*. North-Holland, 1974.

[44] A. Krizhevsky, I. Sutskever and G. E. Hinton. ‘Imagenet classification with deep convolutional neural networks’. In: *Advances in neural information processing systems*. 2012, pp. 1097–1105.

[45] M. Maalej, V. Paisante, P. Ramos, L. Gonnord and F. Pereira. ‘Pointer Disambiguation via Strict Inequalities’. In: *Code Generation and Optimisation*. Austin, United States, Feb. 2017. URL: https://hal.archives-ouvertes.fr/hal-01387031.

[46] M. Maalej Kammoun. ‘Low-cost memory analyses for efficient compilers’. Thèse de doctorat, Université Lyon1. PhD thesis. Université Lyon 1, 2017. URL: http://www.theses.fr/2017LYSE1167.

[47] D. Monniaux and L. Gonnord. ‘Cell morphing: from array programs to array-free Horn clauses’. In: *23rd Static Analysis Symposium (SAS 2016)*. Ed. by X. Rival. Static Analysis Symposium. Edimbourg, United Kingdom, Sept. 2016. URL: https://hal.archives-ouvertes.fr/hal-01206882.

[48] M. Moy. ‘Parallel Programming with SystemC for Loosely Timed Models: A Non-Intrusive Approach’. In: *DATE*. Grenoble, France, Mar. 2013, p. 9. URL: https://hal.archives-ouvertes.fr/hal-00761047.

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

[50] V. Paisante, M. Maalej, L. Barbosa, L. Gonnord and F. M. Q. Pereira. ‘Symbolic Range Analysis of Pointers’. In: *International Symposium of Code Generation and Optmization*. Barcelon, Spain, Mar. 2016, pp. 791–809. URL: https://hal.inria.fr/hal-01228928.

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

[52] L.-N. Pouchet. *Polybench: The polyhedral benchmark suite*. 2012. URL: http://www.cs.ucla.edu/~pouchet/software/polybench/.

[53] W. H. Press, S. A. Teukolsky, W. T. Vetterling and B. P. Flannery. 'Numerical recipes in C++'. In: *The art of scientific computing* (2015).

[54] P. Quinton. 'Automatic synthesis of systolic arrays from uniform recurrent equations'. In: *ACM SIGARCH Computer Architecture News* 12.3 (1984), pp. 208–214.

[55] H. Rihani, M. Moy, C. Maiza, R. I. Davis and S. Altmeyer. 'Response time analysis of synchronous data flow programs on a many-core processor'. In: *RTNS*. ACM. 2016, pp. 67–76.

[56] H. Rihani, M. Moy, C. Maïza, R. I. Davis and 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*. RTNS '16. Brest, France: ACM, 2016, pp. 67–76. DOI: 10.1145/2997465.2997472. URL: http://doi.acm.org/10.1145/2997465.2997472.

[57] H. N. W. Santos, I. Maffra, L. Oliveira, F. Pereira and 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, Oct. 2014. URL: http://hal.inria.fr/hal-01006209.

[58] B. da Silva, A. Braeken, E. H. D'Hollander and A. Touhafi. 'Performance modeling for FPGAs: extending the roofline model with high-level synthesis tools'. In: *International Journal of Reconfigurable Computing* 2013 (2013), p. 7.

[59] W. Thies. 'Language and compiler support for stream programs'. PhD thesis. Massachusetts Institute of Technology, 2009.

[60] J. Travis and J. Kring. *LabVIEW for everyone: graphical programming made easy and fun*. Prentice-Hall, 2007.

[61] A. Turjan. 'Compiling Nested Loop Programs to Process Networks'. PhD thesis. Universiteit Leiden, 2007.

[62] N. Ventroux and 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.

[63] S. Verdoolaege. 'Polyhedral Process Networks'. In: Handbook of Signal Processing Systems. Springer, 2010, pp. 931–965.

[64] S. Williams, A. Waterman and D. Patterson. 'Roofline: an insightful visual performance model for multicore architectures'. In: *Communications of the ACM* 52.4 (2009), pp. 65–76.

[65] P. Wilmott. *Quantitative Finance*. Wiley, 2006.