RESEARCH CENTRE

Grenoble - Rhône-Alpes

IN PARTNERSHIP WITH:

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

2021 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 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

A2.2.5. – Run-time systems

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, Starting Faculty Position]
- Yannick Zakowski [Inria, Researcher]

Faculty Members

- Matthieu Moy [Team leader, Univ Claude Bernard, Associate Professor, HDR]
- Laure Gonnord [Univ Claude Bernard, Associate Professor, until Aug 2021, HDR]

PhD Students

- Thais Baudon [École Normale Supérieure de Lyon, from Sep 2021]
- Julien Braine [École Normale Supérieure de Lyon, until Nov 2021]
- Nicolas Chappe [École Normale Supérieure de Lyon, from Sep 2021]
- Julien Emmanuel [Bull, CIFRE]
- Paul Iannetta [École Normale Supérieure de Lyon, until Nov 2021]
- Amaury Maille [École Normale Supérieure de Lyon]
- Hugo Thievenaz [Inria, from Sep 2021]

Interns and Apprentices

- Clement Allain [École Normale Supérieure de Lyon, from Mar 2021 until Jul 2021]
- Thais Baudon [École Normale Supérieure de Lyon, from Feb 2021 until Jul 2021]
- Melvyn Bertolone–Lopez-Serrano [École Normale Supérieure de Lyon, from May 2021 until Jul 2021]
- Peio Borthelle [École Normale Supérieure de Lyon, from Mar 2021 until Aug 2021]
- Alexis Carre [Inria, from Sep 2021]
- Nicolas Chappe [École Normale Supérieure de Lyon, from Mar 2021 until Jun 2021]
- Quentin Corradi [École Normale Supérieure de Lyon, from Feb 2021 until Jun 2021]
- Damien De Montis [Paratools SAS, from Mar 2021 until Sep 2021]
- Ilham Lasfar [École Normale Supérieure de Lyon, from Apr 2021 until Aug 2021]
- Sebastien Michelland [École Normale Supérieure de Lyon, from Oct 2021]
- Valentin Pasquale [École Normale Supérieure de Lyon, from Feb 2021 until Jun 2021]
- Alban Reynaud [École Normale Supérieure de Lyon, from Feb 2021 until Jun 2021]
- Bastien Rousseau [École Normale Supérieure de Lyon, from Sep 2021]
- Alec Sadler [Inria, from May 2021 until Aug 2021]
- Ambre Suhamy [École Normale Supérieure de Lyon, from Mar 2021 until Sep 2021]
- Hugo Thievenaz [Inria, from Feb 2021 until Jul 2021]

Administrative Assistant

- Solene Audoux [Inria]

Visiting Scientist

- Reiner Hähnle [Université Darmstadt, Allemagne, Sep 2021]

External Collaborator

- Laure Gonnord [Institut polytechnique de Grenoble, from Sep 2021, Computer Science Professor at Esisar Eng. School, HDR]

2 Overall objectives

Research 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*. To reach this goal, the CASH team provides efficient analyses and optimizing compilation frameworks for dataflow programming models. These contributions are combined with two other research directions. First, research on foundations of programming language and program analysis provides a theoretical basis for our work. Second, parallel and scalable simulation of hardware systems, combined with high-level synthesis tools, result in an end-to-end workflow for circuit design.

The scientific focus of CASH is on compute kernels and assembly of kernels, and the first goal is to improve their efficient compilation. However the team also works in collaboration with application developers, to understand better the overall need in HPC and design optimizations that are effective in the context of the targeted applications. 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).

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 (we often use the polyhedral model, a powerful theory to optimize loop nests, but it does not support data structures such as lists). Analysis on these programs should be able to detect non licit memory accesses, memory consumption, hotspots, . . . , and to prove functional properties.

Our Approach and methodology. We target a balance between theory and practice: problems extracted from industrial requirements often yield theoretical problems.

On the practical side, the CASH team targets applied research, in the sense that most research topics are driven by actual needs, discussed either through industrial partnership or extracted from available benchmarks.

The theoretical aspects ensure the coherency and the correctness of our approach. We 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 allow us to show that these different tasks will be performed with the same understanding of the program semantics.

Our approach is to cross-fertilize between several communities. For example, the abstract interpretation community provides a sound theoretical framework and very powerful analysis, but these are rarely

applied in the context of optimizing compilation. Similarly, the hardware simulation community usually considers compilers as black-boxes and does not interact with researchers in compilation.

While a global approach links CASH activities and members, we do not plan to have a single unified toolchain where all contributions would be implemented. For example, contributions in the domain of static analysis of sequential programs may be implemented in the LLVM tool, results on dataflow models are applied both in the SigmaC compiler and in the DCC HLS tool, ... This also implies that different activities of CASH target different application domains and potential end-users.

Research directions. The main objectives of the cash team are to provide *scalable and expressive static analysis* and *optimizing parallel compilers*. These directions rely on programming languages and representation of programs in which *parallelism and dataflow* play a crucial role. A central research direction aims at the study of parallelism and dataflow aspects in programming languages, both from a practical perspective (syntax or structure), and from a theoretical point of view (semantics). The CASH team also has *simulation activities* that are both applied internally in CASH, to simulate intermediate representations, and for embedded systems.

3 Research program

3.1 Research direction 1: Parallel and Dataflow Programming Models

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 is 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 [64] 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 [45] or Streamit [60], where the scheduling is fully static, and general communicating networks like KPNs [48] or RVC-Cal [27] 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 [36] that nowadays introduce more and more dataflow primitives to enable data-driven interactions between agents, particularly with *futures* [33, 41].

Formal semantics

Proving the correctness of an analysis or of a program transformation requires a formal semantics of the language considered. Depending on the context, our formalizations may take the form of paper definitions, or of a mechanization inside of a proof assistant. While more time consuming, the latter may ensure in the adequate context some additional trust in the proofs established, as well as a tighter connection to an executable artifact. We have been recently studying in particular the formalization of concurrent and parallel paradigms, under weak memory models notably, by building on top of the interaction tree [67] approach developed for the Coq proof assistant.

Programming models and program transformations.

So far, the programming models designed in this direction allow to express parallelism in novel ways, but don't leverage the optimising compiler transformation introduced in direction 3. Indeed, optimising compilers only provide control over their behavior through extra-language annotations called "pragmas". Since those annotations are outside the language, they do not benefit from abstraction and modularity, and are often brittle. We plan to provide better integration between the optimisation passes of compiler inside the language itself through the use of meta-programming, by presenting optimisations as first class objects which can be applied, composed and manipulated in the language. A first step of this long term project is to investigate how to express loop transformations (developed by polyhedral model approaches) using existing meta-programming constructs.

3.1.1 Expected Impact

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

Medium-term activities. We plan to 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 Research direction 2: Expressive, Scalable and Certified 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.

We are interested both in paper proofs and verified proofs using a proof assistant such as Coq. Formally verified analysis crucially rely on a formal semantics of the programming language the analysis operates on: Yannick Zakowskiprecisely developed recently a new formal semantics in Coq for the sequential fragment of LLVM IR [9], the intermediate representation at the heart of the LLVM compilation infrastructure.

The semantics of Vellvm, which technically relies on Interaction Trees [67], enjoys crucial properties of compositionality and modularity. By leveraging these meta-theoretic properties of the semantics of the language, we believe that the additional objective of formal correctness can be compatible with the objectives of expressivity and scalability of the analyses we wish to develop for LLVM in particular.

The design of formal semantics allows formulating well-foundedness criteria relatively to the language semantics, that we can use 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 [54, 50], as well as tailoring static analyses for specialized applications in compilation [40, 59], Synchronous Dataflow scheduling [58], and extending the polyhedral model to irregular applications [24]. We also have experience in the design of various static verification techniques adapted to different programming paradigms.

Modularity of programming languages Modularity is an essential property of modern programming languages, allowing to assemble pieces of software in a high level and composable fashion. We aim to develop new module systems and tools for large scale ecosystems. A first aspect of this work is to pursue the collaboration with Didier Remy (Inria Cambium) and Jacques Garrigue (University of Nagoya) on designing module systems for ML languages. Gabriel Radanne is working on the formalization and implementation of a new rich module system which can serve as foundation for further experiment on the OCaml module system. A second aspect is to improve the ease of use of large ecosystems. We also work on tools to assist software developers, such as a tool to search functions by types, in a way that scales to complete ecosystems.

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

Medium-term activities. In the context of Paul Iannetta's Phd thesis, we have proposed a *semantic rephrasing* of the polyhedral model and proposed first steps toward and effective "polyhedral-like compilation" for algebraic datastructures like trees. In medium term, we want to extend the applicability of this new model for arbitrary layouts. The must challenging ones are sparse matrices. This activity still relies 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 [51], whose contribution is a sequence of memory analyses inside production compilers. We already began a collaboration with Caroline 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 [44] 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 [28] 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 [35] 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 Research direction 3: Optimizing Program Transformations

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 [62] and rely directly on array dataflow analysis [38]. 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 [39]. 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 [65] 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) [25]. 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 [43]. 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.

Certified compilation We will develop a research direction around the formal proof of compilation passes, and of optimizing program transformations in particular. Although realistic formally verified optimizing compilers are roughly 15 years old, three limitations to the current state of the art are apparent.

First, loop optimizations have been very sparsely tackled, their proof rising difficult semantic issues. We intend on one side to leverage the compositionality of Interaction-Tree-based semantics as used in

Vellvm to improve the situation. An orthogonal axis we wish to explore is the formalization in Coq of the Polyhedral Model, as pioneered in 2021 by Courant and Leroy [34].

Second, parallelism and concurrency have been almost ignored by the verified compilation community. This problem is a major long term endeavor for we first need to develop the appropriate semantic tools. Ludovic Henrio and Yannick Zakowski will work with a master student, Ambre Suhamy, to explore the use of Interaction Trees to model various paradigms for concurrency, paving the long term way to an extension of Vellvm to concurrency.

Third, these proofs are very brittle for they rely on concrete implementation of memory models rather than axiomatizations of those. Ludovic Henrio and Yannick Zakowski will work with a master student, Alban Reynaud, to develop semantic tools to reason formally up-to arbitrary algebras in Coq. One of the core objectives of this project is to prove optimizations at a higher level of abstraction, so that these proofs remain valid by construction under changes in the memory model.

The compiler analyses proposed above 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) [25]. 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 investiguate the points described thereafter.

Roofline model and dataflow-level resource evaluation. Operational intensity must be tuned according to the roofline model. The roofline model [65] 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 [57] 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.

3.3.1 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 plan to evaluate the impact of state-of-the-art polyhedral source-to-source transformations on HLS for FPGA. Our results on polyhedral HLS (DPN [10, 26]) could also be a good starting point for this purpose. We will give a particular focus to memory layout transformations, easier to implement as a source level transformation. Then, we will tackle control optimizations throught the adaptation of loop tiling to HLS constraints.

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.

Also, we plan to explore how polyhedral transformations might *scale on larger applications*, typically those found in deep-learning algorithms. We will investigate how the regularity of polyhedral kernels can be exploited to infer general affine transformations from a few offline execution traces. This is the main goal of the *PolyTrace exploratory action*, started on 2021 in collaboration with Waseda University.

We will first target offline memory allocation, an important transformation used in HLS and generally in automatic parallelization.

Finally, 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. 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.

Also, we plan to systematize the definition of scalable polyhedral compilers using extrapolation from offline traces. Both theoretical and applied research are required to reach this goal. The research strategy consists in studying several instances (memory allocation, scheduling, etc). Then, in producing the theoretical ingredients to reach a general methodology of conception.

3.4 Research direction 4: Simulation and 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) [53] in SystemC [47] 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 3.3 (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 [31]. We focus on semi-automatic approaches where the programmer only needs to make minor modifications of programs to get significant speedups. We already obtained results in the joint PhD (with Tanguy

Sassolas) of Gabriel Busnot with CEA-LIST. The research targets parallelizing SystemC heterogeneous simulations, extending SCale [63], which is very efficient to simulate parallel homogeneous platforms such as multi-core chips. We removed the need for manual address annotations, which did 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 can now parallelize simulation running with a full software stack including Linux.

We are also working with Bull/Atos on HPC interconnect simulation, using SimGrid [37]. Our goal is to allow simulating an application that normally runs on a large number of nodes on a single computer, and obtain relevant performance metrics.

3.4.1 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.

3.4.2 Scientific Program

Medium-term activities. We started working on the "heterogeneous" aspect of simulations with an approach allowing changing the level of details in a simulation at runtime.

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 [63] from CEA-LIST, some compute-intensive parts delegated to sc-during [52] from Matthieu Moy, and a co-simulation with external physical solvers using SystemC-MDVP [29] 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.

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 [55]. The irregular versions can be found in [56]. Numerical kernels used in quantitative finance [66] are also good candidates, e.g., finite difference and Monte-Carlo simulation.

The medium-size applications we target are streaming algorithms [27], scientific workflows [61], and also the now very rich domain of deep learning applications [49]. 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 [30] 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.

Ludovic Henrio followed the "Atelier Sciences Environnements Sociétés Inria 2021" (atelier Sens) organized by Eric Tannier in June 2021. Then, for the voluntary Cash members, he has animated an atelier Sens during the Cash seminar in October 2021.

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, Dcc was enhanced with an automated scheduler. Specifically, we have implemented an affine loop tiling algorithm and the related affine scheduler optimized for pipelined datapath. We also pursue the debugging of symbolic parallelization.

Publication: hal-03143777

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.

Release Contributions: v4.2: - Piece affine array contraction (extension of the successive minima algorithm) - Fast projection heuristics

News of the Year: This year, Christophe implemented significant features in PoCo. First, a piecewise affine extension of the successive minima array contraction algorithm was added. Also, a set of heuristics for faster polyhedron projection were implemented.

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

Contact: Christophe Alias

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

Contact: Ludovic Henrio

6.1.5 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

Contact: Christophe Alias

Participants: Christophe Alias, Guillaume Iooss, Sanjay Rajopadhye

Partner: Colorado State University

6.1.6 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

Contact: Christophe Alias

Participant: Christophe Alias

6.1.7 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

Contact: Yannick Zakowski

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

Partner: University of Pennsylvania

6.1.8 vaphor

Name: Verification of Programs with Horn Clauses

Keyword: Program verification

Functional Description: Program to horn clauses horn clauses with arrays abstraction

Contact: Laure Gonnord

Partner: Vérimag

6.1.9 Data Abstraction

Name: Data Abstraction

Keywords: Static analysis, Program verification, Propositional logic

Functional Description: The tool is an element of a static program (or other) verification process which is done in three steps:

1. Transform the verification problem into Horn clauses, perhaps using MiniJavaConverter or SeaHorn 2. Simplify the Horn clauses using data abstraction (this tool). 3. Solve the Horn clauses using a Horn solver such as Z3

Contact: Laure Gonnord

Partner: Vérimag

6.1.10 S4BXI

Keywords: Simulation, HPC, Network simulator

Functional Description: S4BXI is a simulator of the Portals4 network API. It is written using SimGrid's S4U interface, which provides a fast flow-model. More specifically, this simulator is tuned to model as best as possible Bull's hardware implementation of portals (BXI interconnect)

URL: https://s4bxi.julien-emmanuel.com/docs/

Contact: Julien Emmanuel

Partner: Bull - Atos Technologies

6.1.11 llvm-pass

Name: LLVM Pass analyzer

Keyword: Compilation

Functional Description: tool suite to work on llvm passes

URL: https://github.com/llvmpass/llvm-passes

Contact: Laure Gonnord

6.1.12 kut

Keywords: Polyhedral compilation, Task scheduling

Scientific Description: Task-level parallelism is usually exploited by a runtime scheduler, after tasks are mapped to processing units by a compiler. Kut is the compiler part of a compilation-centric runtime scheduling strategy. With kut, the tasks are partitioned so as to guide the runtime scheduling. Kut inputs an HPC kernel written in C and outputs a C with predicates helping the dynamic decision of the scheduler.

Functional Description: Task-level parallelism is usually exploited by a runtime scheduler, after tasks are mapped to processing units by a compiler. Kut is the compiler part of a compilation-centric runtime scheduling strategy. With kut, the tasks are partitioned so as to guide the runtime scheduling. Kut inputs an HPC kernel written in C and outputs a C with predicates helping the dynamic decision of the scheduler.

News of the Year: 2019: version 1

Publication: hal-02421327

Contact: Christophe Alias

Participants: Christophe Alias, Samuel Thibault, Laure Gonnord

6.1.13 ribbit

Keywords: Compilation, Pattern matching, Algebraic Data Types

Functional Description: Ribbit is a compiler for pattern languages with algebraic data types which is parameterized by the memory representation of types. Given a memory representation, it generates efficient and correct code for pattern matching clauses.

URL: https://gitlab.inria.fr/cash/ribbit

Contact: Gabriel Radanne

6.1.14 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

6.1.15 adtr

Name: ADT Rewriting language

Keywords: Compilation, Static typing, Algebraic Data Types, Term Rewriting Systems

Functional Description: ADTs are generally represented by nested pointers, for each constructors of the algebraic data type. Furthermore, they are generally manipulated persistently, by allocating new constructors.

ADTr allow representing ADTs in a flat way while compiling a pattern match-like construction as a rewrite on the memory representation. The goal is to then use this representation to optimize the rewriting and exploit parallelism.

URL: https://github.com/Drup/adtr

Publication: hal-03355377

Contact: Gabriel Radanne

Participants: Gabriel Radanne, Paul Iannetta, Laure Gonnord

6.1.16 dowsing

Keywords: Static typing, Ocaml

Functional Description: Dowsing is a tool to search function by types. Given a simple OCaml type, it will quickly find all functions whose types are compatible.

Dowsing works by building a database containing all the specified libraries. New libraries can be added to the database. It then builds an index which allow to quickly answer to requests.

URL: https://github.com/Drup/dowsing/

Publication: hal-03355381

Contact: Gabriel Radanne

Participants: Gabriel Radanne, Laure Gonnord

6.1.17 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/

Contact: Gabriel Radanne

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

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 Research direction 1: Parallel and Dataflow Programming Models

7.1.1 Flexible Synchronization for Parallel Computations.

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. Last year we proposed PromisePlus, a synchronization tool allowing tasks to synchronize on chunks of arrays with a granularity configurable by the programmer. We presented these results at the FSEN 2021 conference [14].

This year, we worked on a parametrisation of FIFO queues that are used to stream information between processes. We proposed an extension of classical FIFO queues where the granularity of synchronization can vary at runtime. Preliminary results are promising but additional work is necessary before publishing them.

7.1.2 An Optimised Flow for Futures: From Theory to Practice.

Participants: Nicolas Chappe, Ludovic Henrio, Amaury Maillé, Matthieu Moy, Hadrien Renaud.

A future is an entity representing the result of an ongoing computation. A synchronisation with a "get" operation blocks the caller until the computation is over, to return the corresponding value. When a computation in charge of fulfilling a future delegates part of its processing to another task, mainstream languages return nested futures, and several "get" operations are needed to retrieve the computed value (we call such futures "control-flow futures"). Several approaches were proposed to tackle this issues: the

"forward" construct, that allows the programmer to make delegation explicit and avoid nested futures, and "data-flow explicit futures" which natively collapse nested futures into plain futures. This paper supports the claim that data-flow explicit futures form a powerful set of language primitives, on top of which other approaches can be built. We prove the equivalence, in the context of data-flow explicit futures, between the "forward" construct and classical "return" from functions. The proof relies on a branching bisimulation between a program using "forward" and its "return" counterpart. This result allows language designers to consider "forward" as an optimisation directive rather than as a language primitive. Following the principles of the Godot system, we provide a library implementation of control-flow futures, based on data-flow explicit futures implemented in the compiler. This small library supports the claim that the implementation of classical futures based on data-flow ones is easier than the opposite. Our benchmarks show the viability of the approach from a performance point of view.

The idea of Data-Flow Explicit Future is not new: it has been proposed by Ludovic Henrio in [46], and a partial implementation and theoretical study was done in [42]. An implementation of data-flow explicit futures in the Encore language was made but not published during the M2 internship of Amaury Maillé. We performed performance analysis and compared several implementation during the internship of Hadrien Renaud, and proposed an optimization together with its proof of correctness during the internship of Nicolas Chappe. These three internships were concluded by a journal publication [7].

7.1.3 Locally abstract globally concrete semantics

Participants: Ludovic Henrio, Reiner Hähnle, Einar Broch Johnsen, Crystal Chang Din, Lizeth Tapia Tarifa.

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 and 2021, partly because of Covid restrictions but the visit of Reiner Hähnle in yhe Cash team allosed us to progress on the subject and to prepare a follow-up relating scheduling and LAGC.

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).

7.1.4 PNets: Parametrized networks of automata

Participants: Ludovic Henrio, Quentin Corradi, Eric Madelaine, Rabéa Ameur Boulifa.

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 refinement theory for open pNets. This work is realized with Eric Madelaine (INRIA Sophia-Antipolis), Rabéa Ameur Boulifa (Telecom ParisTech) and Quentin Corradi who did part of his internship in the Cash team.

An article that follows the internship of Quentin Corradi is being written.

7.1.5 A Survey on Verified Reconfiguration

Participants: Ludovic Henrio, Helene Coullon, Frederic Loulergue, Simon Robillard.

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 the article is under major revision. A new version will be submitted in Spring 2022.

7.1.6 A Survey on Parallelism and Determinacy

Participants: Ludovic Henrio, Laure Gonnord, Lionel Morel, 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 has been submitted in Spring 2021, and we are preparing our author response after a first round of review. This work involves three members of the CASH team as well as an external collaborator: Lionel Morel (Insa de Lyon)

7.1.7 Verified Compilation Infrastructure for Concurrent Programs

Participants: Nicolas Chappe, Ludovic Henrio, Matthieu Moy, Ambre Suhamy, Yannick Zakowski.

The objective of this research direction is to provide semantic and reasoning tools for the formalization of concurrent programs and the verification of compilers for concurrent languages. In particular, we want to apply these results to the design of verified optimizing compilers for parallel high-level languages. We wish to proceed in the spirit of the approach advocated in Vellvm [9]: compositional, modular, executable monadic interpreters based on Interaction Trees [67] are used to specify the semantics of the language, in contrast with more traditional transition systems. Proving correct optimizations for such concurrent languages naturally requires new proof techniques that we need to design as well. This research direction is at its early stages, but we have promising early results. This year we have the following new achievements:

- During the internship of Ambre Suhamy, we have designed a concurrent semantics, based on an extension of interaction trees, for CCS (Calculus of communicating systems). We proved preliminary results over this semantics.
- Based on this first investigation, we have started the design of a major extension of interaction trees (dubbed "ctrees") for internal non-determinism — allowing in praticular to encode concurrency. The formalisation and ground theory has been designed, leading to the definition of notions of bisimulations over these constructions. We will investigate its application to a couple of use cases in 2022. This line of work is a collaboration with Steve Zdancewic and Paul He from University of Pennsylvania.
- Nicolas Chappe has started his PhD thesis under the supervision of Yannick Zakowski, Matthieu Moy, and Ludovic Henrio. The PhD is entitled “Toward a verified compilation infrastructure for concurrent programs: Non-sequential paradigms in Vellvm”. The first investigations conducted during the PhD has been dedicated to the design of semantics (based on ctrees) for concurrent programs supporting weak memory models.

7.2 Research direction 2: Expressive, Scalable and Certified Analyses

7.2.1 Data Abstraction: A General Framework to Handle Program Verification of Data Structures

Participants: Julien Braine, Laure Gonnord, David Monniaux.

Proving properties on programs accessing data structures such as arrays often requires universally quantified invariants, e.g., "all elements below index i are nonzero". In our paper accepted at SAS 2021 [11] (long version in [17], we propose a general data abstraction scheme operating on Horn formulas, into which we recast previously published abstractions. We show that our instantiation scheme is relatively complete: the generated purely scalar Horn clauses have a solution (inductive invariants) if and only if the original problem has one expressible by the abstraction.

7.2.2 Search functions by types

Participants: Gabriel Radanne, Laure Gonnord, Clement Allain.

Sometimes, we need a function so deeply that we have to go out and search for it. How do we find it? Sometimes, we have clues: a function which manipulates `list` is probably in the `List` module ... but not always! Sometimes, all we have is its functionality: doing the sum of a list of integers. Unfortunately, search by functionality is difficult.

Rittri proposed an approximation: use the *type* of the function as a key to search through libraries: in our case, `int list -> int`. To avoid stumbling over details such as the order of arguments, he proposed to use matching *modulo type isomorphism* – a notion broader than syntactic equality. Unfortunately, algorithms for unification modulo type isomorphisms are extremely costly (at best NP). An exhaustive search would not be usable during programming in practice.

In this research direction, we investigate how to scale search by types. For this purpose, we developed new algorithm technique similar to indexes used in databases, but appropriate for keys following a rich language of types. Early result have been published in ML2021 [15] and we have developed a prototype, Dowsing 6.1.16, implementing these ideas.

7.2.3 A new module system for OCaml

Participants: Clement Blaudeau, Didier Remy, Gabriel Radanne.

Modularity is a key tool for building reusable and structured code. While the OCaml module system is known for being expressive and powerful, specifying its behavior and formalizing its properties has proven to be hard. This research direction aims to develop a comprehensive description of a generative subset of the OCaml module system (called the *source system*), with a focus on the *signature avoidance problem*. By extending the signature syntax with existential types (inspired by F_ω), we describe an alternate system (called *canonical*) with a simpler set of judgments that both solves the issues of the source description and provides formal guarantees through elaboration into F_ω. We show that the canonical system is more expressive than the source one and state at which conditions canonical typing derivations can be translated back into the source typing. As a middle point between the path-based representation of OCaml and the formal description of F_ω, the canonical presentation is a framework which we believe could support numerous features (applicative functors, transparent ascription, module aliases, ...) and could serve as a basis for a redefinition of module system of an industry-grade language as OCaml.

7.3 Research direction 3: Optimizing Program Transformations

7.3.1 Optimizing compiling for tree-like structures

Participants: Paul Ianetta, Laure Gonnord, Gabriel Radanne.

Trees are used everywhere, yet their internal operations are nowhere as optimized as arrays are. This work is in the continuity of the research on cache-oblivious algorithms for trees. It investigate the properties of structural transformations on trees via two directions.

First, we studied how to develop structural low-level operations to efficiently manipulate trees in-place and how to optimized and parallelize such operations [18]. This yielded an optimized implementation to manipulate AVL trees 6.1.14.

Following this study, we developed a more general framework that aims to decompose and optimize arbitrary transformation on trees by expressing them as rewrite rules on Algebraic Data Types. In this approach, the traditional semantic of functional programs with pattern matching is reinterpreted as optimized, in-place, imperative loop nests. These results were published in GPCE2021 [13] and implemented in a prototype compiler 6.1.15.

7.3.2 Parallel rewriting operations on Trees

Participants: Thaïs Baudon, Carsten Fuhs, Laure Gonnord.

During the M2 internship of Thaïs Baudon, partially funded by the CODAS's ANR project, we studied the problem of scheduling programs with complex data structures with the point of view of term rewriting. As related work, we show that although scheduling operations on efficient data structures apart from arrays should have a great impact on performance, there is nearly no work on this topic. However, some related work from the rewriting community gives us great insights about the link between termination and scheduling. Our contributions as a first step toward full efficient compilation of programs with inductive data structures are 1. first algorithms for this parallel evaluation, 2. a code generation algorithm to generate efficient parallel evaluators, 3. a prototype implementation and first experimental results

Part of this work, namely the proposition of a novel parallel complexity notion was published in the WST workshop [16].

7.3.3 Memory optimizations for Algebraic Data Types

Participants: Thaïs Baudon, Gabriel Radanne, Laure Gonnord.

In the last few decades, Algebraic Data Types (ADT) have emerged as an incredibly effective tool to model and manipulate data for programming. Additionally, ADTs could provide numerous advantages for optimizing compilers, as the rich declarative description could allow them to choose the memory representation of the types.

Initially, ADTs were mostly present in functional programming languages such as OCaml and Haskell. Such GC-managed functional languages generally use uniform memory representation which prohibit agressive optimisations of the representation of ADTs. However, ADTs are now present in many different languages, notably Scala and Rust, which permit such optimizations.

The goal of this research direction is to investigate how to represent terms of Algebraic Data Types and how to compile pattern matching efficiently. We aim to develop a generic compilation framework which accomodate arbitrarely complex memory representation for terms, and to provide news ways to optimize the representation of ADTs. A prototyper compiler has been implemented 6.1.13.

7.3.4 Vellvm: Verified LLVM

Participants: Calvin Beck, Bastien Rousseau, Irene Yoon, Yannick Zakowski, Steve Zdancewic.

We develop, in collaboration with the University of Pennsylvania, a formally verified in Coq compilation infrastructure based on LLVM, dubbed Vellvm 6.1.7. Compared to other existing verified compilation framework, we define the semantics of the languages we consider as monadic interpreters built on top of the Interaction Trees framework. This approach brings us benefits in terms of modularity, compositionality and executability, as well as leads to an equational mode of reasoning to establish refinements. The following major achievements have taken place this year:

- We have published the description of our new infrastructure at ICFP'21 [9].
- A major redefinition of the memory model, led by Calvin Beck and accounting for the necessary finite view of the memory imposed by the presence of pointer to integer casts in LLVM IR, has been started. Most of it is finished, we are working on validating the new model by proving optimizations corrects and will submit an article about it during the year 2022.
- Reasoning about the various monadic interpreters involved in such semantics is challenging. A major process of abstraction and generalization of the equational theory expected from these monads, and a zoo of implementations of these theories, has been developed — an effort led by Irene Yoon. These results will be submitted as an article during the year 2022.
- Spiral is a compilation framework for the generalization of efficient low level code for numerical computations. HELIX is a formalization in Coq of part of this framework that Vadim Zaliva has developped during his PhD. We have proved correct a back end of the HELIX compilation chain down to Vellvm. The full chain will be described in a journal paper during the year 2022.
- As part of his internship, Bastien Rousseau has developed a set of verified combinators to build control flow graphs compositionally and anonymously. By providing in this way a DSL to write program transformations over LLVM IR, we obtain for free a set of well-formedness properties over the generated code. Furthermore, each combinator coming with a semantic characterestic equation, we obtain compositional reasoning principles. We plan on complementing this work with its dual: a DSL of patterns to deconstruct a graph into its construction as combinators.

7.3.5 Verified Abstract Interpreters as Monadic Interpreters

Participants: Laure Gonnord, Sébastien Michelland, Yannick Zakowski.

In the realm of verified compilation, one typically wants to verify the static analyzes used by the compiler. In existing works, the analysis is typically written as a fuel-based pure function in Coq and verified against the semantics described as a transition system. The goal of this research is to develop the tools and reasoning principles to transfer these ideas to a context where the semantics of the language is defined as a monadic interpreters built on Interaction Trees.

During his internship, Sébastien Michelland has developed a first promising prototype, establishing a highly modular framework to build and prove correct such analyses. He has instantiated his result on a toy Imp language, and is now aiming at instantiating it on a toy assembly language. We plan on submitting an article describing these first results, before considering the application of the technique to Vellvm.

7.3.6 A Complete Compilation Chain for Polyhedral High-Level Synthesis

Participants: Christophe Alias, Alexandru Plesco.

High-Level Synthesis tools lacks high-level optimizations. This work shows the feasibility of using the polyhedral model to design the HLS tool itself. We propose a complete compilation chain based on 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.

This work has been accepted for publication in the CC'21 conference [10]. The compiler has been completely implemented 6.1.1, 6.1.2. It is now used in the XTREMLOGIC startup.

7.3.7 Enhancing the Polyhedral Model with Parametric Tiling

Participants: Guillaume Iooss, Christophe Alias, Sanjay Rajopadhye.

For polyhedral programs, parametric tiling in its full generality is known to be non-linear, breaking the mathematical closure properties of the polyhedral model. Most compilation tools therefore either perform fixed size tiling, or apply parametric tiling in only the final, code generation step. We introduce monoparametric tiling, a restricted parametric tiling transformation. We show that, despite being parametric, it retains the closure properties of the polyhedral model. We first prove that applying monoparametric partitioning (i) to a polyhedron yields a union of polyhedra with modulo conditions, and (ii) to an affine function produces a piecewise-affine function with modulo conditions. We then use these properties to show how to tile an entire polyhedral program. Our monoparametric tiling is general enough to handle tiles with arbitrary tile shapes that can tessellate the iteration space (e.g., hexagonal, trapezoidal, etc). This enables a wide range of polyhedral analyses and transformations to be applied.

This work has been accepted for publication in the international journal of parallel programming [8]. We also provide a complete implementation 6.1.3.

7.3.8 Affine Multibanking for High-Level Synthesis

Participants: Christophe Alias, Matthieu Moy.

High-level circuit compilers (high-level synthesis, HLS) are able to translate a C program to an FPGA circuit configuration. Unlike software parallelization, there is no operating system to place the computation and the memory at runtime. All the parallelization decisions must be done at compile time. In this work, we address the compilation of data placement under parallelism and resource constraints. We propose an HLS algorithm able to partition the data across memory banks, so parallel access will target distinct banks to avoid data transfer serialization. Our algorithm is able to minimize the number of banks and the maximal bank size.

The first results are still preliminary, but promising [19]. We are working on extending the experimental validation to more general polyhedral kernels.

7.3.9 Towards a Trace-Based Polyhedral Model

Participants: Hugo Thievenaz, Keiji Kimura, Christophe Alias.

In this work, we defend the iconoclast idea that polyhedral optimizations might be computed without those operations, simply by applying a lightweight analysis on a few off-line execution traces. The main intuition being that, since polyhedral transformations are expressed as affine mappings, only a few points are required to infer the general mapping. Our hope is to compute those points from a few off-line execution traces. We focus on array contraction, a well known technique to reallocate temporary arrays thanks to affine mappings so the array size is reduced. We describe a liveness algorithm from an execution trace, and another to compute the maximum number of variables alive alongside a dimension, from which we get our scalar modular mappings. We show that a simple interpolation allow to infer the modulo mapping.

This work is funded by the polytrace Inria exploratory action. Our results are still preliminary, but promising [22]. We are currently working on the cases which require piece-wise affine mappings.

7.3.10 A Polyhedral Approach for Scalar Promotion

Participants: Alec Sadler, Christophe Alias, Hugo Thievenaz.

Memory accesses are a well known bottleneck whose impact might be mitigated by using properly the memory hierarchy until registers. In this paper, we address array scalarization, a technique to turn temporary arrays into a collection of scalar variables to be allocated to registers. We revisit array scalarization in the light of the recent advances of the polyhedral model, a general framework to design optimizing program transformations. We propose a general algorithm for array scalarization, ready to be plugged in a polyhedral compiler, among other passes. Our scalarization algorithm operates on the polyhedral intermediate representation. In particular, our scalarization algorithm is parametrized by the program schedule possibly computed by a previous compilation pass. We rely on schedule-directed array contraction and we propose a loop tiling algorithm able to reduce the footprint down to the available amount of registers on the target architecture. Experimental results confirm the effectiveness and the efficiency of our approach.

This is a preliminary, but promising work [21]. We are working on extending the tiling model to handle more dependence patterns.

7.3.11 A Polyhedral Approach for Auto-Parallelization using a Distributed Virtual Machine

Participants: Damien De Montis, Jean-Baptiste Besnard, Christophe Alias.

As parallel systems have to undergo an unprecedented transition towards more parallelism and hybridization, we propose to discuss the consequences on programming models. In particular, MPI and OpenMP may face some complexity barriers due to the added complexity required by such hardware. We propose to build from the ground up a new way to program a parallel system, relying both on a distributed runtime, unifying multiple nodes in a coherent ensemble, and on advanced tools from the Polyhedral model. We first describe the Distributed Virtual Machine (DVM) runtime establishing a data-flow environment suitable to the polyhedral transformations. We present and identify what we have seen as key components in such a system, transposing loop nests to a distributed set of machines thanks to a pragma notation combined with an automatic tiling approach. Eventually, while presenting results we open the discussion of remaining challenges and some potential mitigation.

This is a preliminary work, which will serve as basis for a collaboration with Paratools [20]. An ANR project is under submission around these topics.

7.4 Research direction 4: Simulation and Hardware

7.4.1 Standard-compliant parallel SystemC simulation of loosely-timed transaction level models: From baremetal to Linux-based applications support. Integration

Participants: Matthieu Moy, Gabriel Busnot, Tanguy Sassolas, Nicolas Ventroux.

To face the growing complexity of System-on-Chips (SoCs) and their tight time-to-market constraints, Virtual Prototyping (VP) tools based on SystemC/TLM2.0 must get faster while maintaining accuracy. However, the ASI SystemC reference implementation remains sequential and cannot leverage the multiple cores of modern workstations. In this paper, we present SCale 2.0, a new implementation of a parallel and standard-compliant SystemC kernel, reaching unprecedented simulation speeds. By coupling a parallel SystemC kernel with shared resources access monitoring and process-level rollback, we can preserve SystemC atomic thread evaluation while leveraging the available host cores. We also generate process interaction traces that can be used to replay any simulation deterministically for debug purpose. Evaluation on baremetal applications shows ×15 speedup compared to the ASI SystemC kernel using 33 host cores reaching speeds above 2300 Million simulated Instructions Per Second (MIPS). Challenges related to parallel simulation of full software stack with modern operating systems are also addressed with speedup reaching ×13 during recording run and ×24 during the replay run.

This is a continuation from the earlier conference article [32]. We extended the approach to support simulation of Linux-based applications, and published the results in [6].

7.5 S4BXI: the MPI-ready Portals 4 Simulator

Participants: Julien Emmanuel, Matthieu Moy, Ludovic Henrio, Grégoire Pichon.

We present a simulator for High Performance Computing (HPC) interconnection networks. It models Portals 4, a standard low-level API for communication, and it allows running unmodified applications that use higher-level network APIs such as the Message Passing Interface (MPI). It is based on SimGrid, a framework used to build single-threaded simulators based on a cooperative actor model. Unlike existing tools like SMPI, we rely on an actual MPI implementation, hence our simulation takes into account MPI's implementation details in the performance. This paper also presents a case study using the BullSequana eXascale Interconnect (BXI) made by Atos, which highlights how such a simulator can help design space exploration (DSE) for new interconnects.

This is a continuation of [37] where we proposed the low-level simulator for the Portals 4 protocol. We extended it to support MPI and published it in the MASCOTS conference [12].

8 Bilateral contracts and grants with industry

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

Participants: Matthieu Moy, Ludovic Henrio, Julien Emmanuel.

8.2 Discussions with the Aniah startup on circuit verification

Participants: Matthieu Moy, Ludovic Henrio, Gabriel Radanne.

The CASH team is in discussion with the Aniah startup to collaborate on hierarchical circuit verification. This should lead to an official contract in 2022.

8.3 Discussions with ParaTools on parallelization of HPC kernels

Participants: Christophe Alias.

Christophe Alias is in discussion with ParaTools to collaborate on coarse-grain parallelization of HPC kernels. A join ANR project with Inria Bordeaux (Storm team) has been submitted.

8.4 Discussions with Kalray on code optimization for deep-learning applications

Participants: Christophe Alias.

Christophe Alias is in discussion with Kalray to collaborate on compiler optimizations for Deep Learning algorithms. A last year engineer student will start this year, targetting a CIFRE contract.

8.5 CAVOC Project with Inria/Nomadic Labs

Participants: Guilhem Jaber, Gabriel Radanne, Laure Gonnord.

This project aims to develop a *sound and precise static analyzer* for OCaml, that can catch large classes of bugs represented by uncaught exceptions. It will deal with both user-defined exceptions, and built-in ones used to represent *error behaviors*, like the ones triggered by `failwith, assert`, or a match failure. Via "assert-failure" detection, it will thus be able to check that invariants annotated by users hold. The analyzer will reason *compositionally* on programs, in order to analyze them at the granularity of a function or of a module. It will be *sound* in a strong way: if an OCaml module is considered to be correct by the analyzer, then one will have the guarantee that no OCaml code interacting with this module can trigger uncaught exceptions coming from the code of this module. In order to be *precise*, it will take into account the abstraction properties provided by the type system and the module system of the language: local values, abstracted definition of types, parametric polymorphism. The goal being that most of the interactions taken into account correspond to typeable OCaml code.

This project is part of the partnership between Inria and Nomadic Labs, and lead by Guilhem Jaber, from the Inria Team Galinette.

9 Partnerships and cooperations

9.1 International initiatives

9.1.1 Inria associate team not involved in an IIL or an international program

CAPESA

Title: Characterization of Software Evolution with Static Analyses

Duration: 2020 ->

Coordinator: Sébastien Mosser (mosser.sebastien@uqam.ca)

Partners:

- Université du Québec À Montréal

Inria contact: Laure Gonnord

Summary:

9.1.2 Participation in other International Programs

PHC Aurora: SLAS

Title: Synchronous Languages meet Asynchronous Semantics

Partner Institution(s):
- university of Bergen, Norway
- university of Oslo, Norway

Date/Duration: 2020 (extended to 2021 due to Covid)

9.2 International research visitors

9.2.1 Other International Collaborations

POLYTRACE Inria exploratory action

Title: Polyhedral Compilation from Execution Traces

Partner Institutions :

- Waseda University, Tokyo, Japan

Duration: 2021 -> 2024

Coordinator: Christophe Alias

9.2.2 Visits of international scientists

Other international visits to the team

Reiner Hähnle

Status Professor

Institution of origin: University of Darmstadt

Country: Germany

Dates: September 2021

Context of the visit: Collaboration with Ludovic Henrio on semantics of languages for parallel programming.

Mobility program/type of mobility: Invited professor

10 Dissemination

10.1 Promoting scientific activities

10.1.1 Scientific events: organisation

- Together with Emmanuelle Saillard, Kevin Martin and Frédéric Dubrowski, Laure Gonnord and Ludovic Henrio animate the remote "CLAP" seminar

General chair, scientific chair

- Laure Gonnord was chair of the EJCP Summer school, 2021 remote edition.

Member of the organizing committees

- Laure Gonnord co organized the GDR GPL National Days 2021, remote edition.

Member of the conference program committees

- Christophe Alias was member of the PC of COMPAS 2021.
- Laure Gonnord belongs to the PC of PLDI 2022 and SOAP 2022
- Ludovic Henrio was member of the PC of HLPP2021, ICE2021, and FM 2021.
- Gabriel Radanne was member of the PC of PEPM2021 and of the AEC of CC2021.
- Yannick Zakowski was member of the PC of CPP 2021 and belongs to the PC of ICFP 2022.

Member of the conference steering committees

- Ludovic Henrio is member of the steering committe for ICE workshops.

10.1.2 Journal

Reviewer - reviewing activities

- Christophe Alias has been reviewer for MICPRO and PARCO.
- Ludovic Henrio has been reviewer for IJPP and LMCS.
- Gabriel Radanne has been reviewer for JFP and TIOT.

10.1.3 Teaching

- Bachelor (“Licence”):
  - Christophe Alias, INSA CVL, 3A STI, "Compilation": 26h ETD (CM+TD).
  - Thaïs Baudon Université Claude Bernard Lyon1, LIFASR4 (Architecture des Ordinateurs), L2 Informatique : 16h TD + 24h TP;
  - Thaïs Baudon Université Claude Bernard Lyon1, LIFAP1 (Algorithmique et programmation en C) L1 Informatique : 27h TP.
  - Paul Iannetta Université Claude Bernard Lyon1, L2 informatique, LIFAP2 : Algorithmique et programmation fonctionnelle: 18h TP
  - Julien Braine Université Claude Bernard Lyon1, Bases de Données avancées, L3 Informatique : 32h TD;
  - Laure Gonnord Université Claude Bernard Lyon1, L3 LIFASR5 Systèmes : 20h
  - Laure Gonnord Esisar engineer school, Valence, 3A, C programming labs TP 30h
  - Matthieu Moy co-responsabilité de la mention informatique de la licence de sciences UCBL.
  - Gabriel Radanne Université Claude Bernard Lyon1, L3, LIFProject (Ecandrement de projets étudiants): 16h.
- Master:
  - Christophe Alias, ENS de Lyon, Préparation à l’agrégation d’informatique, "Compilation": 10h CM.
  - Christophe Alias, INSA CVL, 4A STI, "Optimisation des applications embarquées": 26h ETD (CM+TD).
  - Christophe Alias, Laure Gonnord and Yannick Zakowski: ENSL, M2 IF course (sept-nov 2021) "Advanced static analyses for compilation", CR 14.
  - Damien Pous and Yannick Zakowski: ENSL, M2 IF course (sept-nov 2021) "Program Verification with Coinduction and Proof Assistants", CR 16.

- Paul Iannetta: Université Claude Bernard Lyon1, M1 informatique, MIF01 : "Gestion de projet et génie logiciel": 12h TP, 3h TD
- Laure Gonnord: Esisar engineer school, Valence, 4A, "Compilation course" CM+TD 20h
- Laure Gonnord: Esisar engineer school, Valence, 4A, "Database course" CM+TD 25h
- Matthieu Moy: Université Claude Bernard Lyon1, M1 informatique, MIF01 : "Gestion de projet et génie logiciel": 12hCM, 12h TP, 3h TD
- Matthieu Moy: Université Claude Bernard Lyon1, M1 informatique, MIF08 : "Compilation et Traduction de programmes": 7h30CM, 15h TP, 7h30TD
- Gabriel Radanne, Ludovic Henrio, and Nicolas Chappe: ENSL, M1: "Compilation et Analyse de Programmes" 28h CM, 24h TP.
- Ludovic Henrio: University of Nice Sophia Antipolis, M2 Ubinet. "an algorithmic approach to distributed systems"3h30 CM+TD

10.1.4 Supervision

- Hugo Thievenaz's PhD thesis is supervised by Christophe Alias and Keiji Kimura (Waseda University, Japan).
- Julien Braine's PhD thesis is supervised by Laure Gonnord and David Monniaux (Verimag, Grenoble)
- Paul Iannetta's PhD thesis is supervised by Laure Gonnord, Gabriel Radanne and Lionel Morel (Insa de Lyon)
- Thaïs Baudon's PhD thesis is supervised by Laure Gonnord and Gabriel Radanne
- Julien Emmanuel's PhD thesis is supervised by Matthieu Moy and Ludovic Henrio
- Amaury Maillé's PhD thesis is supervised by Ludovic Henrio and Matthieu Moy
- Nicolas Chappe's PhD thesis is supervised by Yannick Zakowski, Matthieu Moy and Ludovic Henrio

10.1.5 Juries

- Christophe Alias was a reviewer in the PhD committee of Salwa Kobeissi, University of Strasbourg. Advisor: Philippe Clauss, Title: "Speculative Rewriting of Recursive Programs as Loop Candidates for Efficient Parallelization and Optimization Using an Inspector-Executor Mechanism".
- Yannick Zakowski was an examiner for the practical programming exam of the concours of the ENS.
- Christophe Alias was in the admission committee for the second concours of ENS de Lyon as an examiner.
- Laure Gonnord was in the PhD commitee of TODO
- Matthieu Moy is in the PhD jury of Nicolas Leclaire, TIMA, supervised by Stéphane Mancini. Title: Methodology for Code-Optimization of Memory Data-Layouts for High-Performance-System Architectures with Complex Memory Hierarchies. Report submited in december 2021, defense planned in january 2022.

10.2 Popularization

10.2.1 Internal or external Inria responsibilities

- Laure Gonnord was member of the "Inria Commission d'Evaluation" until sept 2021
- Laure Gonnord is member of the GDR GPL board and in charge of the associated Young researcher school

11 Scientific production

11.1 Major publications

[1] C. Alias and A. Plesco. 'Data-Aware Process Networks'. In: CC 2021 - 30th ACM SIGPLAN International Conference on Compiler Construction. Virtual, South Korea: ACM, 2nd Mar. 2021, pp. 1–11. DOI: 10.1145/3446804.3446847. URL: https://hal.inria.fr/hal-03143777.

[2] G. Busnot, T. Sassolas, N. Ventroux and M. Moy. 'Standard-compliant parallel SystemC simulation of loosely-timed transaction level models: From baremetal to Linux-based applications support'. In: *Integration, the VLSI Journal* 79 (July 2021), pp. 23–40. DOI: 10.1016/j.vlsi.2020.12.006. URL: https://hal.archives-ouvertes.fr/hal-03487607.

[3] N. Chappe, L. Henrio, A. Maillé, M. Moy and H. Renaud. 'An Optimised Flow for Futures: From Theory to Practice'. In: *The Art, Science, and Engineering of Programming* 6.1 (15th July 2021), pp. 1–41. DOI: 10.22152/programming-journal.org/2022/6/3. URL: https://hal.inria.fr/hal-03440766.

[4] 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, 2019, pp. 1–28. URL: https://hal.archives-ouvertes.fr/hal-02302214.

[5] Y. Zakowski, C. Beck, I. Yoon, I. Zaichuk, V. Zaliva and S. Zdancewic. 'Modular, compositional, and executable formal semantics for LLVM IR'. In: *Proceedings of the ACM on Programming Languages* 5.ICFP (22nd Aug. 2021), pp. 1–30. DOI: 10.1145/3473572. URL: https://hal.archives-ouvertes.fr/hal-03525711.

11.2 Publications of the year

International journals

[6] G. Busnot, T. Sassolas, N. Ventroux and M. Moy. 'Standard-compliant parallel SystemC simulation of loosely-timed transaction level models: From baremetal to Linux-based applications support'. In: *Integration, the VLSI Journal* 79 (July 2021), pp. 23–40. DOI: 10.1016/j.vlsi.2020.12.006. URL: https://hal.archives-ouvertes.fr/hal-03487607.

[7] N. Chappe, L. Henrio, A. Maillé, M. Moy and H. Renaud. 'An Optimised Flow for Futures: From Theory to Practice'. In: *The Art, Science, and Engineering of Programming* 6.1 (15th July 2021), pp. 1–41. DOI: 10.22152/programming-journal.org/2022/6/3. URL: https://hal.inria.fr/hal-03440766.

[8] G. Iooss, C. Alias and S. Rajopadhye. 'Monoparametric Tiling of Polyhedral Programs'. In: *International Journal of Parallel Programming* 49 (18th Mar. 2021), pp. 376–409. DOI: 10.1007/s10766-021-00694-2. URL: https://hal.inria.fr/hal-02493164.

[9] Y. Zakowski, C. Beck, I. Yoon, I. Zaichuk, V. Zaliva and S. Zdancewic. 'Modular, compositional, and executable formal semantics for LLVM IR'. In: *Proceedings of the ACM on Programming Languages* 5.ICFP (22nd Aug. 2021), pp. 1–30. DOI: 10.1145/3473572. URL: https://hal.archives-ouvertes.fr/hal-03525711.

International peer-reviewed conferences

[10] C. Alias and A. Plesco. 'Data-Aware Process Networks'. In: CC 2021 - 30th ACM SIGPLAN International Conference on Compiler Construction. Virtual, South Korea: ACM, 2nd Mar. 2021, pp. 1–11. DOI: 10.1145/3446804.3446847. URL: https://hal.inria.fr/hal-03143777.

[11] J. Braine, L. Gonnord and D. Monniaux. 'Data Abstraction: A General Framework to Handle Program Verification of Data Structures'. In: SAS 2021 - 28th Static Analysis Symposium. Vol. 12913. Lecture notes in computer science. Chicago, United States, 13th Oct. 2021, pp. 215–235. DOI: 10.1007/978-3-030-88806-0_11. URL: https://hal.inria.fr/hal-03321868.

[12] J. Emmanuel, M. Moy, L. Henrio and G. Pichon. 'S4BXI: the MPI-ready Portals 4 Simulator'. In: MASCOTS 2021 - 29th IEEE International Symposium on the Modeling, Analysis, and Simulation of Computer and Telecommunication Systems. Houston, United States: IEEE, 3rd Nov. 2021, pp. 1–8. URL: https://hal.inria.fr/hal-03366573.

[13] P. Iannetta, L. Gonnord and G. Radanne. 'Compiling pattern matching to in-place modifications'. In: GPCE 2021 - 20th International Conference on Generative Programming: Concepts & Experiences. Chicago & Virtual, United States, 17th Oct. 2021. DOI: 10.1145/3486609.3487204. URL: https://hal.archives-ouvertes.fr/hal-03355377.

[14] A. Maillé, L. Henrio and M. Moy. 'Promise Plus: Flexible Synchronization for Parallel Computations on Arrays'. In: *Lecture Notes in Computer Science*. FSEN 2021 - 9th IPM International Conference on Fundamentals of Software Engineering. Tehran, Iran, 19th May 2021, pp. 1–7. URL: https://hal.archives-ouvertes.fr/hal-03143269.

Conferences without proceedings

[15] C. Allain, G. Radanne and L. Gonnord. 'Isomorphisms are back!: Smart indexing for function retrieval by unification modulo type isomorphisms'. In: ML 2021 - ML Workshop. Virtual, France, 26th Aug. 2021, pp. 1–3. URL: https://hal.archives-ouvertes.fr/hal-03355381.

[16] T. Baudon, C. Fuhs and L. Gonnord. 'Parallel Complexity of Term Rewriting Systems'. In: WST 2021 - 17th International Workshop on Termination. Virtual, France, 16th July 2021, pp. 1–6. URL: https://hal.archives-ouvertes.fr/hal-03418400.

Reports & preprints

[17] J. Braine, L. Gonnord and D. Monniaux. *Data Abstraction: A General Framework to Handle Program Verification of Data Structures*. RR-9408. Inria Grenoble Rhône-Alpes; VERIMAG UMR 5104, Université Grenoble Alpes, France; LIP - Laboratoire de l'Informatique du Parallélisme; Université Lyon 1 - Claude Bernard; ENS Lyon, 1st May 2021, pp. 1–29. URL: https://hal.inria.fr/hal-03214475.

[18] P. Iannetta, L. Gonnord and G. Radanne. *Parallelizing Structural Transformations on Tarbres*. RR-9405. ENS Lyon, CNRS & INRIA, 26th Apr. 2021, p. 21. URL: https://hal.inria.fr/hal-03208466.

[19] I. Lasfar, C. Alias, M. Moy, R. Neveu and A. Carré. *Affine Multibanking for High-Level Synthesis*. RR-9440. Inria - Research Centre Grenoble – Rhône-Alpes, 20th Dec. 2021. URL: https://hal.inria.fr/hal-03481328.

[20] D. de Montis, J.-B. Besnard and C. Alias. *A Polyhedral Approach for Auto-Parallelization using a Distributed Virtual Machine*. RR-9432. INRIA , LIP - ENS Lyon; Paratools, 25th Oct. 2021, p. 25. URL: https://hal.inria.fr/hal-03402663.

[21] A. Sadler, C. Alias and H. Thievenaz. *A Polyhedral Approach for Scalar Promotion*. RR-9437. Institut National de Recherche en Informatique et en Automatique (INRIA), 25th Nov. 2021, p. 14. URL: https://hal.inria.fr/hal-03449994.

[22] H. Thievenaz, K. Kimura and C. Alias. *Towards Trace-Based Array Contraction*. RR-9442. Inria; Waseda University, 15th Dec. 2021, p. 20. URL: https://hal.inria.fr/hal-03482055.

11.3 Other

Softwares

[23] [SW] M. Belaoucha, C. Alias, D. Barthou and S. Touati, *FADAlib: an open source C++ library for fuzzy array dataflow analysis*, 25th Nov. 2021. LIC: GNU Lesser General Public License v3.0 or later. HAL: ⟨hal-03445991⟩, URL: https://hal.archives-ouvertes.fr/hal-03445991, SWHID: ⟨swh:1:dir:fc7481ee438316b9ce5b273ca894114bf658d3d9;origin=https://hal.archives-ouvertes.fr/hal-03445991;visit=swh:1:snp:518f2d28a2d2a1ad15ee2f630b40be3e24a0f8b1;anchor=swh:1:rel:488f5aa5aaa21fc92f24f0f7c9b571e56f1325ec;path=/⟩.

11.4 Cited publications

[24] 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.

[25] 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.

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

[27] 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.

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

[29] 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.

[30] 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.

[31] 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.

[32] 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, Jan. 2020, pp. 1–6. URL: https://hal.archives-ouvertes.fr/hal-02416253.

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

[34] N. Courant and X. Leroy. 'Verified Code Generation for the Polyhedral Model'. In: *Proceedings of the ACM on Programming Languages* 5.POPL (Jan. 2021), 40:1–40:24. DOI: 10.1145/3434321. URL: https://hal.archives-ouvertes.fr/hal-03000244.

[35] 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.

[36] 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.

[37] 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, Dec. 2020, pp. 1–8. URL: https://hal.archives-ouvertes.fr/hal-02972297.

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

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

[40] 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.

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

[42] 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.

[43] 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.

[44] 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.

[45] 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.

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

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

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

[49] 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.

[50] 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.

[51] 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.

[52] 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.

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

[54] 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.

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

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

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

[58] 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.

[59] 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.

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

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

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

[63] 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.

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

[65] 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.

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

[67] L. Xia, Y. Zakowski, P. He, C. Hur, G. Malecha, B. C. Pierce and S. Zdancewic. 'Interaction Trees'. In: *Proceedings of the ACM on Programming Languages* 4.POPL (2020). DOI: 10.1145/3371119.