Proposal for a joint “équipe projet commune” (joint project team).

CASH: Compilation and Analysis, Software and Hardware

December 10, 2018

Name: CASH: Compilation and Analysis, Software and Hardware
Laboratory: Laboratoire d’Informatique du Parallélisme, Lyon, France
Research Institutes (tutelles): CNRS, ENS-Lyon, Inria, UCBL
Inria research field: Algorithmics, Programming, Software and Architecture / Architecture, Languages and Compilation
Scientific leader: Matthieu Moy
Members:

Christophe Alias (Researcher, Inria)
Julien Braine (Ph.D student)
Laure Gonnord (Associate Professor, Univ. Lyon1)
Ludovic Henrio (Researcher, CNRS)
Paul Iannetta (Ph.D student)
Laetitia Lecot (Inria assistant)
Matthieu Moy (Team Leader, Associate Professor, Univ. Lyon1)

Contents

LIP – UMR CNRS / ENS Lyon / UCB Lyon 1 / Inria – 69007 Lyon
E-mail : Christophe.Alias@inria.fr, Laure.Gonnord@univ-lyon1.fr, Ludovic.Henrio@cnrs.fr, Matthieu.Moy@univ-lyon1.fr

1 Introduction

1.1 Scientific Context

The advent of parallelism in supercomputers and in more classical end-user computers increases the need for high-level code optimization and improved compilers.

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

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

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

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

These issues are considered as one of the main challenges in the Hipeac roadmap [21] which, among others, cites the two major issues:

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

1.2 Scientific Foundations and Transversal Topics

1.2.1 Parallelism and Dataflow

Parallelism based on dataflow is one way to tackle energy-efficiency and get control on data movement. The dataflow formalism expresses a computation on an infinite number of values, that can be viewed as successive values of a variable during time. A dataflow program is structured as a set of *processes* that communicate values through *buffers*.

Examples of dataflow languages include the synchronous languages Lustre [31] and Signal [35]. SigmaC [9] is a dataflow oriented programming language dedicated to HPC. The DPN representation [4] (data-aware process network) is an example of a dataflow intermediate representation for a parallelizing compiler.

A dataflow program or representation can be implemented in several ways: as software running either on a parallel general-purpose architecture or on accelerators like GPUs or many-cores, or as hardware implementation, possibly running on reconfigurable chips (FPGAs).

Dataflow formalisms are a transverse topic of this proposal. We plan to use them both as programming languages and as intermediate representations within compilers. However, we will not a priori restrict ourselves to dataflow applications (we also consider approaches to optimize languages dedicated to GPU like CUDA and OpenCL code for example). We believe that the dataflow formalism is a good starting point and a convergence point for all the members of the team.

1.2.2 Automatic Parallelization and Polyhedral Model

The main challenge of automatic parallelization is to reason about the computation. With the polyhedral model, the iterations of compute-intensive loop kernels are abstracted away as integral points satisfying affine constraints (i.e., set of points in a polyhedron). This way, precise, *iteration-level*, compiler analyses and optimizations can be performed thanks to geometric operations and linear optimization (to quote a few: dataflow analysis [22], scheduling [14], memory allocation [1]).

Historically, the polyhedral model focused on loop kernels with *Affine Control Loops* (ACL). An ACL program satisfies the following properties: (i) only `for` loops and `if` conditions operating on arrays (ii) the loop bounds, conditions and array accesses are affine functions of surrounding loop counters and structure parameters (typically the array size). This should *not* be viewed as a limitation, but only as one possible use case. Actually, the polyhedral model could apply to *any* program providing the right *abstraction* of iterations as polyhedra. How to find the right level of abstraction and granularity is a transverse issue addressed in Sections 2.1 and 2.3.

In particular, we plan to explore the extensions of polyhedral techniques to handle irregular programs with `while` loops and complex data structures (such as trees and lists). This raises many issues. We cannot represent finitely all the possible executions traces. Which approximation/representation to choose? Then, how to adapt existing techniques on approximated traces while preserving correctness?

1.3 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*. More precisely, we plan to work on:

1. Definition of dataflow representations of parallel programs that can capture the parallelism at all levels: fine-grain vs. coarse-grain, data & task parallelism, programming language, and intermediate representation (Section 2.1).
2. Scalable and expressive static program analyses. CASH will work on improving the scalability of analyses to allow a global analysis of large-scale programs, and on the expressiveness of analysis to find better program invariants. Analysis will be performed both on the representation defined above and on general programs (Section 2.2).
3. Transformations from and to the dataflow representation, combining traditional tools dedicated to dataflow and specific methods like the polyhedral model (Section 2.3).
4. A high-level synthesis (HLS) tool, built on the above item (instantiated with the particularities of FPGAs) and a code generation tool (Section 2.4). This HLS tool will focus on early stages of compilation and rely on an external tool for the back-end.
5. A parallel and scalable simulation of hardware systems, which, combined with the preceding activity, will result in an end-to-end workflow for circuit design (Section 2.5).

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

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

1.4 Overall Approach

- Cross-fertilization with several scientific communities.
- Case-study driven approach and transfer.

The CASH team targets applied research, in the sense that most research topics are driven by actual needs, discussed either through industrial partnership (see Section 5) or extracted from available benchmarks (e.g. Polybench [45]). We target a balance between theory and practice: problems extracted from industrial requirements often yield theoretical problems.

Our approach is to cross-fertilize between several communities. For example, the abstract interpretation community provide 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 do not interact with researchers in compilation.

Our contributions are therefore expected in several domains. While a global approach links CASH activities and members (see Section 2), we do not plan to have a single unified toolchain where all contributions will 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.

1.5 Application Domains

- Objective: analyze, compile, generate code, simulate HPC programs for various platforms.
- Unified approach but various application domains.
- Hard lock: **parallelism everywhere** $\rightsquigarrow$ study various applications from different domains to validate our models and tools.
- Typical applications: **HPC kernels**, **streaming and dataflow applications**.

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 [45]. The irregular versions can be found in [46]. Numerical kernels used in quantitative finance [58] are also good candidates, e.g., finite difference and Monte-Carlo simulation.

The medium-size applications we target are streaming algorithms [6], scientific workflows [53], and also the now very rich domain of deep learning applications [34]. We explore the possibilities of writing (see Section 2.1) and compiling (see Section 2.3) applications using a dataflow language. As a first step,

we will target dataflow programs written in SigmaC [9] 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 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 will also work 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. In particular, we plan to collaborate with the Avalon team (LIP/Inria) that is expert in HPC applications and their design.

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 (2.1) and analyses (2.2) 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 (2.5) 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.

2 Research Directions

Figure 1 illustrates the organization of the research directions of our proposal (corresponding to the 5 research objectives stated in Section 1.3); the illustration relies on the flow of transformation steps performed to compile and simulate a source program. Each red rounded number represents one of the research directions presented below (e.g., static analysis – 2 – is detailed in Section 2.2 below). We see at the same time that static analysis plays a central role to improve and contribute to many transformations, but also that our intermediate dataflow model (defined in Section 2.1) and the transformations from and to this model (described in Section 2.3) play a crucial role in the CASH proposal. We also see that we aim at taking as input both C programs, consisting of big kernels, typically a few hundred lines, but also bigger applications designed as a dataflow parallel program. Finally, the figure illustrates the three execution platforms we target, represented by the three last research directions below: general purpose compilation, HLS specific aspects (HLS compilation also relies on some results of general purpose compilation), and simulation.

2.1 Definition of dataflow representations of parallel programs

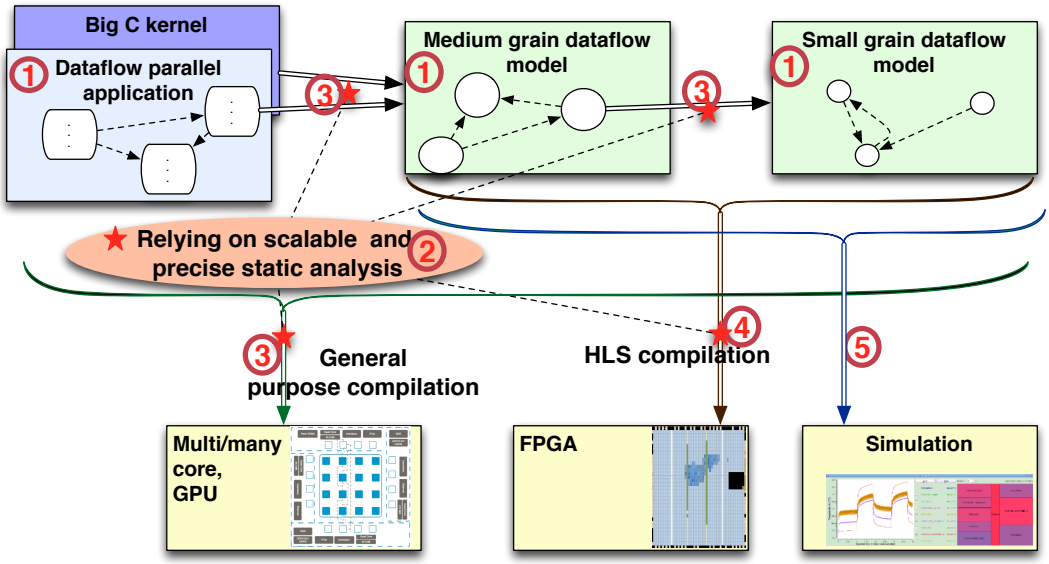

Figure 1: Organisation of research directions

- Objective: provide a unified intermediate representation for dataflow programs.
- Hard lock: **diversity of purpose** ⇝ design formal representation for proving correctness of optimization, and practical model on which efficient optimizations can be implemented.

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 will rely 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 ourself to a strict definition of dataflow languages and, more generally, we are interested in the parallel languages in which dataflow synchronization plays a significant role.

Intermediate dataflow model. The intermediate dataflow model is a representation of the program that is adapted for optimization and scheduling. It will be obtained from the analysis of a (parallel or sequential) program and should at some point be used for compilation. The dataflow model must specify precisely its semantics and parallelism granularity. It must also be analyzable with polyhedral techniques, where powerful concepts exist to design compiler analysis, e.g., scheduling or resource allocation. Polyhedral

Process Networks [56] 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 [31] or Streamit [52], where the scheduling is fully static, and general communicating networks like KPNs [33] or RVC-Cal [6] 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 will be 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 will express parallelism in a dataflow way instead of the classical communication-oriented dependencies. The higher-level more declarative point of view will make 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 [13] that nowadays introduce more and more dataflow primitives to enable data-driven interactions between agents, particularly with *futures* [16, 25].

2.1.1 Expected Impact

Consequently, the impact of this research direction is both the usability of our representation for static analyses and optimizations performed in Sections 2.2 and 2.3, and the usability of its semantics to prove the correctness of these analyses.

2.1.2 Scientific Program

Short-term and ongoing activities. We obtained preliminary experimental [3, 4, 26] and theoretical [32] results, exploring several aspects of dataflow models. The next step is to define accurately the intermediate dataflow model and to study existing programming and execution models:

- Define our medium-grain dataflow model. So far, a modular Polyhedral Process Networks appears as a natural candidate but it may need to be extended to be adapted to a wider range of applications. Precise semantics will have to be defined for this model to ensure the articulation with the activities discussed in Section 2.3.
- Study precisely existing dataflow languages, their semantics, their programmability, and their limitations.

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

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

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

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

2.2 Expressivity and Scalability of Static Analyses

- Objective: design safe and efficient compilers for general programs.
- Hard lock: **scalability** $\rightsquigarrow$ design lowcost analyses tailored to state-of-the-art optimizations.

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 2.1 and 2.3 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 2.1 will allow us to adapt the analysis to the characteristics of the language, and prove that such an adaptation is well founded. This approach will be applicable both on the source language and on the intermediate representation.

Such wellfoundedness criteria relatively to the language semantics will first be used to design our analyses, and then to study which extensions of the languages can be envisioned and analyzed safely, and which extensions (if any) are difficult to analyze and should be avoided. Here the correct identification of a core language for our formal studies (see Section 2.1) 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 [43, 36], as well as tailoring static analyses for specialized applications in compilation [24, 50], Synchronous Dataflow scheduling [49], and extending the polyhedral model to irregular applications [2]. We also have experience in the design of various static verification techniques adapted to different programming paradigms.

2.2.1 Expected impact

The long-term expected impact of this work is the significantly widened applicability of various tools or compilers related to parallelization. We consider two main goals: extend the applicability of heavyweight analysis and parallelization techniques to a wider class of programs, and leverage the applicability of *abstract domains* so that they could be used as analysis passes for client analyses/optimizations in state-of-the-art compiler infrastructures such as LLVM.

We target both analysis for optimization and analysis to detect bugs, or prove their absence (numerical overflows in sequential code, parallel bugs, proving program termination, detecting transient errors, ...). We rely on the polyhedral model and on abstract interpretation, both of which are already used for this purpose.

In a longer term, we expect to be able to design abstract domains tailored for application and platform, and apply them to domain specific languages compiled and run on multicore systems as well as specific machines, without reinventing complete toolchains.

2.2.2 Scientific Program

Short-term and ongoing activities. Together with Paul Iannetta and Lionel Morel (INSA/CEA LETI), we are currently working on the *semantic rephrasing* of the polyhedral model [28]. The objective is to clearly redefine the key notions of the polyhedral model on general flowchart programs operating on arrays, lists and trees. We reformulate the algorithms that are performed to compute dependencies in a more semantic fashion, i.e. considering the program semantics instead of syntactical criteria. The next step is to express classical scheduling and code generation activities in this framework, in order to overcome the classical syntactic restrictions of the polyhedral model.

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

At the same time, we plan to continue to work on scaling static analyses for general purpose programs, in the spirit of Maroua Maalej's PhD [37], whose contribution is a sequence of memory analyses inside production compilers. We already began a collaboration with Sylvain Collange (PACAP team of IRISA Laboratory) on the design of static analyses to optimize copies from the global memory of a GPU to the block kernels (to increase locality). In particular, we have the objective to design specialized analyses but with an explicit notion of cost/precision compromise, in the spirit of the paper [30] 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 [7] 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 [18] 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.

2.3 Compiling and Scheduling Dataflow Programs

- Objective: design an end-to-end compiler/scheduler for middle-sized dataflow intensive applications.
- Hard locks: **granularity** $\rightsquigarrow$ combining tools from the polyhedral model and dataflow paradigms.

In this part, we propose to design the compiler analyses and optimizations for the *medium-grain* dataflow model defined in section 2.1. 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 [54] and rely directly on array dataflow analysis [22]. 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 [23]. 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 [57] 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) [4]. 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 [29]. 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.

2.3.1 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.

2.3.2 Scientific Program

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

Together with Lionel Morel (INSA/CEA LETI), we currently work on the evaluation of the practical advantage of combining the dataflow paradigm with the polyhedral optimization framework. We empirically build a proof-of-concept tooling approach, using existing tools on existing languages [27]. We combine dataflow programming with polyhedral compilation in order to enhance program parallelization by leveraging both inter-agent parallelism and intra-agent parallelism (i.e., regarding loop nests inside agents). We evaluate the approach practically, on benchmarks coming from image transformation or neural networks, and the first results demonstrate that there is indeed a room for further improvement.

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

We plan to investigate in the following directions:

- Investigate how polyhedral analysis extends on modular dataflow programs. For instance, how to modularize polyhedral scheduling analysis on our dataflow programs?

- Develop a proof of concept and validate it on linear algebra kernels (SVD, Gram-Schmidt, etc.).

- Explore various areas of applications from classical dataflow examples, like radio and video processing, to more recent applications in deep learning algorithmic. This will enable us to identify some potential (intra and extra) agent optimization patterns that could be leveraged into new language idioms.

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

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

2.4 HLS-specific Dataflow Optimizations

- Objective: design a hardware optimizing compiler from our dataflow representation.
- Hard lock: **memory bandwith** $\rightsquigarrow$ integrate additional HLS constraints such as communication costs.

The compiler analyses proposed in section 2.3 do not target a specific platform. In this part, we propose to leverage these analysis to develop source-level optimizations for high-level synthesis (HLS).

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

Hence, CASH will focus on the *front-end* pass and the *intermediate representation*. Low-level *back-end* techniques are not in the scope of CASH. Specifically, CASH will leverage the dataflow representation developed in Section 2.1 and the compilation techniques developed in Section 2.3 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) [4]. 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 [57] 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 [48] 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.

2.4.1 Expected Impact

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

2.4.2 Scientific Program

Short-term and ongoing activities. The HLS compiler designed in the CASH team currently extracts a fine-grain parallel intermediate representation (DPN [4, 3]) from a sequential program. We will not write a back-end that produces code for FPGA but we need to provide C programs that can be fed into existing C-to-FPGA compilers. However we obviously need an end-to-end compiler for our experiments. One of the first task of our HLS activity is to develop a DPN-to-C code generator suitable as input to an existing HLS tool like Vivado HLS. The generated code should exhibit the parallelism extracted by our compiler, and allow generating a final circuit more efficient than the one that would be generated by our target HLS tool if ran directly on the input program. Source-to-source approaches have already been experimented successfully, e.g. in Alexandru Plesco's PhD [44].

Medium-term activities. Our DPN-to-C code generation will need to be improved in many directions. The first point is the elimination of redundancies induced by the DPN model itself: buffers are duplicated to allow parallel reads, processes are produced from statements in the same loop, hence with the same control automaton. Also, multiplexing uses affine constraints which can be factorized [5]. We plan to study how these constructs can be factorized at C-level and to design the appropriate DPN-to-C translation algorithms.

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

Long-term activities. The DPN-to-C compiler will open new research perspectives. We will explore the roofline model accuracy for different applications by playing on DPN parameters (tile size). Unlike the classical roofline model, the peak performance is no longer assumed to be constant, but decreasing with operational intensity [19]. Hence, we expect a *unique* optimal set of parameters. Thus, we will build a DPN-level cost model to derive an interval containing the optimal parameters.

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

2.5 Simulation of Hardware

- Objective: design effective simulations for our low-level representations of programs, combined with precise models of hardware.
- Hard lock: **various simulation paradigms** $\rightsquigarrow$ integrate heterogenous (and black-box) components in a unique setting.

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) [42] in SystemC [41] 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 2.1 (dataflow models and programs) and 2.4 (application to FPGA). Indeed, some of the parallelization techniques we develop in CASH could apply to SystemC/TLM programs. Conversely, a complete design-flow based on HLS needs fast system-level simulation: the full-system usually contains both hardware parts designed using HLS, handwritten

hardware components, and software.

We will 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 [10]. We focus on semi-automatic approaches where the programmer only needs to make minor modifications of programs to get significant speedups.

2.5.1 Expected Impact

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

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

2.5.2 Scientific Program

Short-term and ongoing activities. We started the joint PhD (with Tanguy Sassolas) of Gabriel Busnot with CEA-LIST. The research targets parallelizing SystemC heterogeneous simulations. CEA-LIST already developed SCale [55], which is very efficient to simulate parallel homogeneous platforms such as multi-core chips. However, SCale cannot currently load-balance properly the computations when the platform contains different components modeled at various levels of abstraction. Also, SCale requires manual annotations to identify accesses to shared variables. These annotations are given as address ranges in the case of a shared memory. This annotation scheme does not work when the software does non-trivial memory management (virtual memory using a memory management unit, dynamic allocation), since the address ranges cannot be known statically. We started working on the "heterogeneous" aspect of simulations with an approach allowing changing the level of details in a simulation at runtime, and started tackling the virtual and dynamic memory management problem by porting Linux on our simulation platform.

We also started working on an improved support for simulation and debugging of the DPN internal representation of our parallelizing compiler (see Section 2.3). A previous quick experiment with simulation was to generate C code that simulates parallelism with POSIX-threads. While this simulator greatly helped debug the compiler, this is limited in several ways: simulations are not deterministic, and the simulator does not scale up since it would create a very large number of threads for a non-trivial design.

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

Medium-term activities. Several research teams have proposed different approaches to deal with parallelism and heterogeneity. Each approach targets a specific abstraction level and coding style. While we do

not hope for a universal solution, we believe that a better coordination of different actors of the domain could lead to a better integration of solutions. We could imagine, for example, a platform with one subsystem accelerated with SCale [55] from CEA-LIST, some compute-intensive parts delegated to sc-during [40] from Matthieu Moy, and a co-simulation with external physical solvers using SystemC-MDVP [8] from LIP6. We plan to work on the convergence of approaches, ideally both through point-to-point collaborations and with a collaborative project.

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

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

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

3 Research group

Matthieu Moy is the current leader of the CASH "équipe centre". He is the proposed leader of the "equipe projet commune".

- **Matthieu Moy**. `https://matthieu-moy.fr`, assistant professor at Université Claude Bernard, Lyon 1 (UCBL). Matthieu received his PhD degree in Computer Science from Grenoble INP in 2005. During his PhD, he applied formal verification on models of Systems-on-a-Chip written in SystemC, as part of a collaboration with STMicroelectronics. After he joined Verimag/Ensimag as an assistant professor, he continued working with STMicroelectronics on various topics including formal verification, compilation techniques for SystemC, modeling of energy consumption and parallel execution of simulations. He presented part of his work on the subject to Collège de France with Laurent Maillet-Contoz from STMicroelectronics[1].
  He also worked on abstract interpretation and real-time calculus. Recently, his main research interest moved to critical systems on many-core architecture, including code generation from synchronous languages and timing analysis. He obtained his Habilitation (HDR) in 2014 from Grenoble INP, and became the leader of the "Synchrone" team in Verimag in 2015. He co-supervised 4 full PhDs (including 2 as main supervisor), and is now co-supervising 3 PhDs (2 as main supervisor).
- **Christophe Alias**. `http://perso.ens-lyon.fr/christophe.alias`, research associate (*Chargé de recherche*) at Inria. Christophe received his PhD degree in Computer Science from University of Versailles in 2005. His research focus on program analysis and compiler optimizations for high-performance computing, with a strong emphasis on polyhedral compilation techniques. He worked on many subjects around HPC compilers, including array SSA, vectorization, theory of loop tiling, data transfer optimization, buffer allocation, and dataflow models for high-level synthesis. He

[1] `http://www.college-de-france.fr/site/gerard-berry/seminar-2014-01-29-17h30.htm`

co-supervised 2 PhD students: Alexandru Plesco on the compilation of data transfers for high-level synthesis, and Guillaume Iooss on semantic tiling. In 2011, he has co-founded with Cédric Bastoul the IMPACT international workshop which is now the annual reference event for the polyhedral compilation community. In 2014, he has co-founded with Alexandru Plesco the XtremLogic startup, which exploits the parallelizing compiler he wrote to synthesize circuits for FPGA.

- **Laure Gonnord**. `http://laure.gonnord.org/pro/`, assistant professor at Université Claude Bernard, Lyon 1 (UCBL). Laure received her PhD degree in computer science from the University Joseph Fourier (Grenoble), in 2007. She has been an assistant professor at the University of Lille, then at UCBL. Her main research interests lie in the design of static analyses, with emphasis on the automatic synthesis of numerical invariants and application in compilation (scheduling) and termination proofs. She has experience in the development of static analyses for synchronous programs. She coordinates the French Compilation Community (`http://compilfr.ens-lyon.fr/`). She obtained her Habilitation in 2017 from UCBL. She co-supervised 1 full PhD, and is from September 2018 co-supervising 2 PhD students.
- **Ludovic Henrio**. `https://lhenrio.github.io/`, research associate (CR1) at CNRS. Ludovic received his PhD degree from the University of Nice Sophia-Antipolis in 2003 and his habilitation thesis from the University of Nice Sophia-Antipolis in 2012. Ludovic Henrio was the scientific leader of the SCALE project at I3S (Sophia-Antipolis) and joined the LIP laboratory and the CASH team in October 2018. His research interests mostly focus on the formal study of programming languages for distributed systems, and on the application of formal methods to the verification of distributed applications and systems. In particular he worked on the theory of programming languages for distributed objects and components, but also on the application of formal methods to the static verification of distributed systems. He has experience in the semantics of programming languages and its use to assert the correctness of static analyses and runtime optimizations. He (co-)supervised 8 PhD students and is currently supervising 1 PhD student hosted at Huawei Technologies.

Participation Matrix

The expected participation of members is depicted in Table 1. This participation matrix is indicative, we expect research directions to cross-fertilize and hence, the participation of members to vary in the future. Directions 2.1, 2.2 and 2.3 are the most transverse hence all members participate; directions 2.4 and 2.5 are more specific, which explains that not all members participate in it.

	2.1	2.2	2.3	2.4	2.5
Christophe Alias	10 %	10 %	30 %	40 %	10 %
Laure Gonnord	20 %	60 %	20 %		
Ludovic Henrio	50 %	30 %	20 %		
Matthieu Moy	20 %	10 %	10 %	10 %	50 %

Table 1: Involvement of people of the team (Indicative percentages of the time of each participant)

4 Positioning with Respect to Other Research Teams

This section summarizes the main research teams working on the same topics as CASH. A more comprehensive list, with more details on the team's activities and how they relate to ours can be found in Appendix A. For the teams closest to CASH, the positioning text was discussed and validated by the

corresponding team leader.

In the following, we use *Laboratory-Team* to refer to teams. For example, the team described in this document is LIP-CASH.

4.1 Research themes and related teams within Inria

The current project proposal intends to make progress in the quest for performance of dataflow programs in particular on FPGAs, building upon the expertise of its members on algorithm design and formal methods.

The research teams that work in these domains in the Inria ecosystem are thus coming from two "themes" inside the "Algorithms, programs, software and architecture" field:

- Architecture, languages, compilation: especially CAMUS and CORSE, and to some extend PACAP.
- Embedded systems and real time: especially SPADES and PARKAS.

Some other Inria teams belonging to other themes have also some shared research themes: ANTIQUE, CELTIQUE, and SOCRATE.

4.2 Large-scale Computing

Many teams target the optimized execution of applications on a large-scale infrastructure. Within the same laboratory as CASH (LIP), Avalon works on programming models and runtimes for distributed and HPC platforms. ROMA works on resource allocation and scheduling, and develops the massively parallel MUMPS software. DATAMOVE focuses on data-management in large-scale heterogeneous infrastructures.

These teams share the same final application domain as CASH, but deal with different issues. CASH focuses on small and medium-size computing kernels, and targets optimizations within one computing node. CASH follows a compilation and a language-based approach, and focuses on static optimizations, that are out of the scope of DATA, ROMA and DATAMOVE.

Outside France several teams work on language approaches for high-performance computing. For example the programming language research group in Uppsala works on actors with some dataflow-oriented synchronization patterns, and the Compiler Technology and Parallel Computing at PELAB in Linköping works on skeleton-based programming to optimize the execution of kernelson GPUs. CASH aims at a lower-level optimization, and target lower-level languages than these approaches, but we aim at collaborating with them.

4.3 High Level Synthesis

We believe that High-Level Synthesis (HLS) is an understudied topic. It is studied in IRISA-CAIRN, TIMA-SLS, but both use an approach different from ours. One contribution of CASH is the cross-fertilization between the compilation and abstract interpretation communities. Also, CASH is more focused on static approaches and does not study the reconfigurability, while CAIRN targets dynamic reconfiguration management, and TIMA-SLS also focuses on task migration and context-switch.

Other teams and projects abroad include the VAST laboratory at University of California Los Angeles, the ROCCC project at University of California Riverside, the Circuits and Systems group at Imperial College or the Panda project at Politecnico di Milano. Most of these projects address all the aspects of circuit synthesis, from the design specification to low-level hardware generation. The goal of CASH is to follow a very focused research on HLS arround dataflow optimizations for HLS. A collaboration is possible with each of these teams.

4.4 Code Optimization and Execution

The added value of CASH in the compilation domain is the static-analysis based approach. Unlike LIG-CORSE and ICube-CAMUS, we do not consider dynamic approaches. We distinguish the analysis phase and the optimization phase and focus on the analysis more than the optimization.

4.5 Abstract Interpretation and Formal Methods

Abstract interpretation was initially studied for program verification, and applied to ensure the safety of programs, i.e. to prove properties of programs. DI-ENS-ANTIQUE, and its ancestors, is the team where abstract interpretation was invented, and applied successfully for example to prove the absence of numerical overflow in the primary flight control software of Airbus planes with the ASTREE tool. Verimag-PACSS has also expertise in abstract interpretation, with the same focus on program verification. We already collaborate with Verimag-PACSS, and started a joint PhD in Fall 2018.

What distinguishes CASH from these teams is the focus on low-cost analysis to allow more optimisation opportunities for compilers. We use abstract interpretation as a sound framework to perform compiler analysis and allow optimization passes to benefit from the invariants we compute. The "compilation" and "abstract interpretation" communities have traditionally been distant, and we want to benefit from the advances made in both.

To ensure sound bases for analysis, well-defined semantics are needed. Teams like LIP-PLUME have an expertise in the domain that will certainly benefit to CASH.

Other teams abroad include teams working on theoritical aspects of abstract interpretation, model-checking and logics as well as their application to software verification: The Automatic Verification Research Group, at University of Oxford (UK), for instance, works on various topics linked with formal verification and correct-by-construction synthesis for software and hardware systems. The IMDEA Software lab, in Madrid (Spain) has expertise on the design of (expensive) logic-based program verification processes, but also on the development of models for cache and memory performance, which might inspire us.

4.6 Dataflow

The dataflow formalism is studied in many contexts. It has successfully been applied in critical embedded systems through the Scade/Lustre language initially developed by the Verimag-Synchrone team. The synchronous dataflow paradigm was also used in similar contexts by DI-ENS-PARKAS and LIG-SPADES, who both used synchronous dataflow for embedded systems, and used it for parallel programming. The KIELER project in Kiel and the Informatics Theory Group in Bamberg also use synchronous languages for real-time embedded systems, but no not target high-performance computing. IRISA-TEA developed the Signal language on similar concepts.

CASH will build on the foundations built by these teams, but will apply the dataflow formalism outside the scope of embedded systems. The real-time guarantees provided by synchronous languages are not relevant in the context of CASH, but on the other hand the application of dataflow to HPC opens the door to many optimizations that are impossible on embedded systems.

CITI-SOCRATE also works on dataflow programming for compute-intensive applications. We collaborate on the subject with Lionel Morel. SOCRATE may also be a user of the tools CASH will produce.

Dataflow is also used in an HLS context by the Compaan project (spin-out from University Leiden), with an approach similar to the one of CASH. We target a more general representation than the Polyhedral Process Networks used in Compaan.

4.7 Simulation of Hardware

Fast simulation of hardware, in particular using SystemC, is a widely studied topic. LIP6-CIAN, CEA-LIST, and TIMA-SLS, are the main French actors in the domain. They all work on techniques to improve simulation speed: parallelism for LIP6-CIAN and CEA-LIST, native simulation of software for TIMA-SLS. The activities of TIMA-SLS are disjoint to ours, but the "simulation" research direction of CASH has a lot in common with LIP6-CIAN and CEA-LIST. Indeed, we already co-supervise a PhD with LIST, and plan to collaborate with CIAN.

Non-french actors include the team of Rolf Drechsler (Bremen), Reiner Leupers (Aachen) and Rainer Dömer (California). Most work by these teams target models written at a lower level than the ones we consider in CASH.

5 Industrial collaborations

The CASH team follows a case-study driven approach. In some cases, well-established benchmarks exist and summarize typical programs encountered in real-life applications. Examples include Polybench/C [45] for the polyhedral model or the LLVM test suite for general programs. Another way to direct our research towards the needs of end-users is industrial collaborations. The primary goal of industrial collaborations for CASH is to keep our research in sync with the current needs and practices of the industry. We hope to collaborate with industrials with both collaborative projects and point-to-point contracts like CIFRE PhDs. One key to the success of this strategy is to recruit good master students, since negotiating a CIFRE PhD showed to be much easier once a good student is identified.

We detail below the relationships we currently have with industrial partners, but we hope to establish new partnerships with other companies in the future.

5.1 XtremLogic

Christophe Alias has a strong relationship with XtremLogic, a startup that he cofounded with one of his former PhD student, Alexandru Plesco. XtremLogic develops a complete hardware compiler from C to Verilog, which was initially developed in the context of the PhD of Alexandru Plesco. Christophe is the sole author of the polyhedral compiler that was partially transferred to XtremLogic in April 2014 under an "Inria exclusive License". Since this date, Christophe is linked to XtremLogic by a "convention de concours scientifique Inria", and spends 20% of his time at XtremLogic as "conseiller scientifique" (scientific consultant).

The CASH team as a whole will strongly benefit from this proximity, but the collaboration will be done respecting our "research" identity and taking into account potential conflicts of interests. We make a clear distinction between the work done by XtremLogic in an industrial context and the research done in CASH.

The proximity with XtremLogic gives the CASH team indirect access to end-users, and allows the team to get a better understanding of the current needs and practical challenges in the area of High-Level Synthesis for High-Performance Computing. This will greatly help the CASH team in its "case-study driven" approach (we do not have access to actual case-studies, which are confidential). From a scientific point of view, even though XtremLogic is a commercial entity, there are still major scientific challenges to solve.

5.2 STMicroelectronics

Matthieu Moy has a long term collaboration with STMicroelectronics since 2002 on simulation of systems-on-chips. Topics studied as part of this collaboration include formal verification of SystemC programs, modeling techniques to ensure faithfulness with respect to functional properties, and extra-functional

properties like time, power-consumption and temperature, and parallelism.

We are still in contact with the team at STMicroelectronics (in particular Laurent Maillet-Contoz and Jérôme Cornet), and plan to continue this collaboration. Hot current topics include parallel and distributed simulation, and relationship between modeling and high-level synthesis (since the same language can be used in the model and the implementation, models are much easier to write).

5.3 Kalray

Laure Gonnord started a collaboration on the SigmaC compiler in 2016, with Lionel Morel (CEA Grenoble). Matthieu Moy has an experience on the MPPA architecture and he collaborates with Kalray on hard-real time topics. The research directions on this topic are detailed in Section 2.3. We started discussions on possible future collaborations on the compilation and analysis of OpenMP programs to optimize the execution of loops for the MPPA architecture.

6 Relevant publications of the team members

Some relevant publications of the future team members.

- Polyhedral Dataflow Programming: a Case Study. Romain Fontaine, **Laure Gonnord** and Lionel Morel, SBAC-PAD 2018.
- Improving Comunication Patterns in Polyhedral Process Networks. **Christophe Alias**, HiP3ES 2018.
- Optimizing Affine Control with Semantic Factorizations. **Christophe Alias** and Alexandru Plesco, ACM Transactions on Architecture and Code Optimization (TACO), 2017.
- Pointer Disambiguation via Strict Inequalities. Maroua Maalej, Vitor Paisante, Pedro Ramos, **Laure Gonnord**, and Fernando Pereira, CGO 2017.
- A Survey of Active Object Languages. Frank De Boer, Vlad Serbanescu, Reiner Hähnle, **Ludovic Henrio**, Justine Rochas, Crystal Din, Einar Broch Johnsen, Marjan Sirjani, Ehsan Khamespanah, Kiko Fernandez-Reyes, Albert Mingkun Yang. ACM Computing Surveys 2017.
- Response Time Analysis of Synchronous Data Flow Programs on a Many-Core Processor. Hamza Rihani, **Matthieu Moy**, Claire Maiza, Robert I. Davis and Sevastian Altmeyer, RTNS 2016.
- Estimation of Parallel Complexity with Rewriting Techniques. **Christophe Alias**, Carsten Fuhs, **Laure Gonnord**, WST 2016.
- Synthesis of ranking functions using extremal counterexamples. **Laure Gonnord**, Gabriel Radanne and David Monniaux, PLDI 2015.
- WCET analysis in shared resources real-time systems with TDMA buses. Hamza Rihani, **Matthieu Moy**, Claire Maiza, and Sebastian Altmeyer, RTNS 2015.
- CART: Constant Aspect Ratio Tiling. Guillaume Iooss, Sanjay Rajopadhye, **Christophe Alias**, and Yun Zou, IMPACT 2014.
- Method of automatic synthesis of circuits, device and computer program associated therewith, **Christophe Alias** and Alexandru Plesco, Patent FR1453308, 2014.
- Optimizing Remote Accesses for Offloaded Kernels: Application to High-Level Synthesis for FPGA, **Christophe Alias**, Alain Darte, and Alexandru Plesco, DATE 2013.
- FPGA-specific synthesis of loop-nests with pipelined computational cores. **Christophe Alias**, Bogdan Pasca, and Alexandru Plesco, MICPRO 36(8), 2012.
- Enhancing the Compilation of Synchronous Dataflow Programs with a Combined Numerical Boolean Abstraction. Paul Feautrier, Abdoulaye Gamatié, and **Laure Gonnord**, CSI Journal of Computing 2012.

- Multi-dimensional Rankings, Program Termination, and Complexity Bounds of Flowchart Programs. **Christophe Alias**, Alain Darte, Paul Feautrier, and **Laure Gonnord**, SAS 2010.
- Asynchronous and Deterministic Objects. Denis Caromel, **Ludovic Henrio**, Bernard Serpette, POPL 2004.

References

[1] Christophe Alias, Fabrice Baray, and Alain Darte. Bee+Cl@k: An implementation of lattice-based array contraction in the source-to-source translator Rose. In *ACM Conf. on Languages, Compilers, and Tools for Embedded Systems (LCTES'07)*, 2007. Cited in section 1.2.2.

[2] Christophe Alias, Alain Darte, Paul Feautrier, and Laure Gonnord. Multi-dimensional rankings, program termination, and complexity bounds of flowchart programs. In *International Static Analysis Symposium (SAS'10)*, 2010. Cited in section 2.2.

[3] Christophe Alias and Alexandru Plesco. Method of automatic synthesis of circuits, device and computer program associated therewith. Patent FR1453308, April 2014. Cited in sections 2.1.2 and 2.4.2.

[4] Christophe Alias and Alexandru Plesco. Data-aware Process Networks. Research Report RR-8735, Inria - Research Centre Grenoble – Rhône-Alpes, June 2015. Cited in sections 1.2.1, 2.1.2, 2.3, 2.4, and 2.4.2.

[5] Christophe Alias and Alexandru Plesco. Optimizing Affine Control with Semantic Factorizations. *ACM Transactions on Architecture and Code Optimization (TACO)* , 14(4):27, December 2017. Cited in section 2.4.2.

[6] Ihab Amer, Christophe Lucarz, Ghislain Roquier, Marco Mattavelli, Mickael Raulet, J-F Nezan, and Olivier Deforges. Reconfigurable video coding on multicore. *Signal Processing Magazine, IEEE*, 26(6):113–123, 2009. Cited in sections 1.5 and 2.1.

[7] Scott Ananian. The static single information form. Master's thesis, MIT, September 1999. Cited in section 2.2.2.

[8] Cédric Ben Aoun, Liliana Andrade, Torsten Maehne, François Pêcheux, Marie-Minerve Louërat, and Alain 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*, pages 278–285. IEEE, 2015. Cited in sections 2.5.2 and A.17.

[9] Pascal Aubry, Pierre-Edouard Beaucamps, Frédéric Blanc, Bruno Bodin, Sergiu Carpov, Loïc Cudennec, Vincent David, Philippe Doré, Paul Dubrulle, Benoît Dupont De Dinechin, François Galea, Thierry Goubier, Michel Harrand, Samuel Jones, Jean-Denis Lesage, Stéphane Louise, Nicolas Morey Chaisemartin, Thanh Hai Nguyen, Xavier Raynaud, and Renaud Sirdey. Extended Cyclo-static Dataflow Program Compilation and Execution for an Integrated Manycore Processor. In *Alchemy 2013 - Architecture, Languages, Compilation and Hardware support for Emerging ManYcore systems*, volume 18 of *Proceedings of the International Conference on Computational Science, ICCS 2013*, pages 1624–1633, Barcelona, Spain, June 2013. Cited in sections 1.2.1 and 1.5.

[10] Denis Becker, Matthieu Moy, and Jérôme Cornet. Parallel Simulation of Loosely Timed SystemC/TLM Programs: Challenges Raised by an Industrial Case Study. *MDPI Electronics*, 5(2):22, 2016. Cited in sections 2.5, A.15, and A.18.2.

[11] Louis Besème. Vers une accélération matérielle de gnuradio. Master's thesis, INSA de Lyon, Département informatique, mars 2015. Encadrants : Tanguy Risset et Florent De Dinechin. Cited in section A.3.2.

[12] Bruno Blanchet, Patrick Cousot, Radhia Cousot, Jérôme Feret, Laurent Mauborgne, Antoine Miné, David Monniaux, and Xavier Rival. A static analyzer for large safety-critical software. In *Programming Language Design and Implementation (PLDI)*, pages 196–207, 2003. Cited in section A.10.2.

[13] Frank De Boer, Vlad Serbanescu, Reiner Hähnle, Ludovic Henrio, Justine Rochas, Crystal Chang Din, Einar Broch Johnsen, Marjan Sirjani, Ehsan Khamespanah, Kiko Fernandez-Reyes, and Albert Mingkun Yang. A survey of active object languages. *ACM Comput. Surv.*, 50(5):76:1–76:39, October 2017. Cited in section 2.1.

[14] Uday Bondhugula, Albert Hartono, J. Ramanujam, and P. Sadayappan. A practical automatic polyhedral parallelizer and locality optimizer. In *Proceedings of the ACM SIGPLAN 2008 Conference on Programming Language Design and Implementation, Tucson, AZ, USA, June 7-13, 2008*, pages 101–113, 2008. Cited in section 1.2.2.

[15] Alban Bourge, Olivier Muller, and Frédéric Rousseau. Automatic high-level hardware checkpoint selection for reconfigurable systems. In *Field-Programmable Custom Computing Machines (FCCM), 2015 IEEE 23rd Annual International Symposium on*, pages 155–158. IEEE, 2015. Cited in section A.15.

[16] Denis Caromel and Ludovic Henrio. *A Theory of Distributed Objects*. Springer-Verlag, 2004. Cited in section 2.1.

[17] Weiwei Chen and Rainer Dömer. Optimized out-of-order parallel discrete event simulation using predictions. In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 3–8. EDA Consortium, 2013. Cited in section A.18.2.

[18] 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)*, pages 238–252, Los Angeles, January 1977. Cited in section 2.2.2.

[19] Bruno da Silva, An Braeken, Erik H D'Hollander, and Abdellah Touhafi. Performance modeling for FPGAs: extending the roofline model with high-level synthesis tools. *International Journal of Reconfigurable Computing*, 2013:7, 2013. Cited in section 2.4.2.

[20] Mickaël Dardaillon, Kevin Marquet, Tanguy Risset, Jérôme Martin, and Henri-Pierre Charles. A new compilation flow for software-defined radio applications on heterogeneous mpsocs. *TACO*, 13(2):19:1–19:25, 2016. Cited in section A.3.1.

[21] M. Duranton, D. Black-Schaffer, K. De Bosschere, and J. Maebe. The hipeac vision for advanced computing in horizon 2020, `https://www.hipeac.net/v13`, 2013. Cited in section 1.1.

[22] Paul Feautrier. Dataflow analysis of array and scalar references. *International Journal of Parallel Programming*, 20(1):23–53, 1991. Cited in sections 1.2.2 and 2.3.

[23] Paul Feautrier. Scalable and structured scheduling. *International Journal of Parallel Programming*, 34(5):459–487, October 2006. Cited in section 2.3.

[24] Paul Feautrier, Abdoulaye Gamatié, and Laure Gonnord. Enhancing the Compilation of Synchronous Dataflow Programs with a Combined Numerical-Boolean Abstraction. *CSI Journal of Computing*, 1(4):8:86–8:99, 2012. RR version = http://hal.inria.fr/hal-00780521/en. Cited in section 2.2.

[25] Kiko Fernandez-Reyes, Dave Clarke, Elias Castegren, and Huu-Phuc Vo. Forward to a promising future. In *Conference proceedings COORDINATION 2018*, 2018. Cited in section 2.1.

[26] Romain Fontaine, Laure Gonnord, and Lionel Morel. Polyhedral Dataflow Programming: a Case Study. In *SBAC-PAD 2018 - 30th International Symposium on Computer Architecture and High-Performance Computing*, pages 1–9, Lyon, France, September 2018. IEEE. Cited in section 2.1.2.

[27] Romain Fontaine, Laure Gonnord, and Lionel Morel. Polyhedral Dataflow Programming: a Case Study. In *International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, pages xxx–xxx. IEEE, September 2018. Cited in section 2.3.2.

[28] Laure Gonnord, Paul Iannetta, and Lionel Morel. Semantic Polyhedral Model for Arrays and Lists. Research Report RR-9183, Inria Grenoble Rhône-Alpes ; UCBL ; LIP - ENS Lyon ; CEA List, June 2018. Cited in section 2.2.2.

[29] Michael 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. Cited in section 2.3.

[30] Oh Hakjoo, Lee Wonchan, Heo Kihong, Yang Hongseok, and Yi 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*, page 49. ACM, 2014. Cited in section 2.2.2.

[31] N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud. The synchronous data flow programming language lustre. *Proceedings of the IEEE*, 79(9):1305–1320, Sep 1991. Cited in sections 1.2.1 and 2.1.

[32] Ludovic Henrio. Data-flow Explicit Futures. Research report, I3S, Université Côte d'Azur, April 2018. Cited in section 2.1.2.

[33] G. Kahn. The semantics of a simple language for parallel programming. In *Information processing*. North-Holland, 1974. Cited in section 2.1.

[34] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012. Cited in section 1.5.

[35] P. Le Guernic, J.-P. Talpin, and J.-C. Le Lann. Polychrony for System Design. *Journal for Circuits, Systems and Computers*, 12(3):261–304, April 2003. Cited in section 1.2.1.

[36] Maroua Maalej, Vitor Paisante, Pedro Ramos, Laure Gonnord, and Fernando Pereira. Pointer Disambiguation via Strict Inequalities. In *Code Generation and Optimisation*, Austin, United States, February 2017. Cited in section 2.2.

[37] Maroua Maalej Kammoun. *Low-cost memory analyses for efficient compilers*. PhD thesis, 2017. Thèse de doctorat, Université Lyon1. Cited in section 2.2.2.

[38] Aline Mello, Isaac Maia, Alain Greiner, and Francois Pecheux. Parallel simulation of systemc tlm 2.0 compliant mpsoc on smp workstations. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2010*, pages 606–609. IEEE, 2010. Cited in section A.17.

[39] David Monniaux and Laure Gonnord. Cell morphing: from array programs to array-free Horn clauses. In Xavier Rival, editor, *23rd Static Analysis Symposium (SAS 2016)*, Static Analysis Symposium, Edimbourg, United Kingdom, September 2016. Cited in section A.10.5.

[40] Matthieu Moy. Parallel Programming with SystemC for Loosely Timed Models: A Non-Intrusive Approach. In *DATE*, page 9, Grenoble, France, March 2013. Cited in section 2.5.2.

[41] Open SystemC Initiative. *IEEE 1666 Standard: SystemC Language Reference Manual*, 2011. Cited in section 2.5.

[42] Open SystemC Initiative (OSCI). *OSCI TLM-2.0 Language Reference Manual*, June 2008. Cited in section 2.5.

[43] Vitor Paisante, Maroua Maalej, Leonardo Barbosa, Laure Gonnord, and Fernando Magno Quintao Pereira. Symbolic Range Analysis of Pointers. In *International Symposium of Code Generation and Optmization*, pages 791–809, Barcelon, Spain, March 2016. Cited in section 2.2.

[44] Alexandru Plesco. *Program Transformations and Memory Architecture Optimizations for High-Level Synthesis of Hardware Accelerators*. Theses, Ecole normale supérieure de lyon - ENS LYON, September 2010. Cited in section 2.4.2.

[45] Louis-Noël Pouchet. Polybench: The polyhedral benchmark suite. *URL: http://www. cs. ucla. edu/~ pouchet/software/polybench/[cited July,]*, 2012. Cited in sections 1.4, 1.5, and 5.

[46] William H Press, Saul A Teukolsky, William T Vetterling, and Brian P Flannery. Numerical recipes in c++. *The art of scientific computing*, 2015. Cited in section 1.5.

[47] Adrien Prost-Boucle, Olivier Muller, and Frédéric Rousseau. Fast and standalone design space exploration for high-level synthesis under resource constraints. *Journal of Systems Architecture*, 60(1):79–93, 2014. Cited in section A.15.

[48] Patrice Quinton. Automatic synthesis of systolic arrays from uniform recurrent equations. *ACM SIGARCH Computer Architecture News*, 12(3):208–214, 1984. Cited in section 2.4.

[49] Hamza Rihani, Matthieu Moy, Claire Maiza, Robert I. Davis, and Sebastian 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, pages 67–76, New York, NY, USA, 2016. ACM. Cited in section 2.2.

[50] Henrique Nazaré Willer Santos, Izabella Maffra, Leonardo Oliveira, Fernando Pereira, and Laure Gonnord. Validation of Memory Accesses Through Symbolic Analyses. In *Proceedings of the 2014 ACM International Conference on Object Oriented Programming Systems Languages And Applications (OOPSLA'14)*, Portland, Oregon, United States, October 2014. Cited in section 2.2.

[51] Manuel Selva, Lionel Morel, and Kevin Marquet. numap: A portable library for low-level memory profiling. In *SAMOS*, pages 55–62. IEEE, 2016. Cited in section A.3.1.

[52] William Thies. *Language and compiler support for stream programs*. PhD thesis, Massachusetts Institute of Technology, 2009. Cited in section 2.1.

[53] Jeffrey Travis and Jim Kring. *LabVIEW for everyone: graphical programming made easy and fun*. Prentice-Hall, 2007. Cited in section 1.5.

[54] Alexandru Turjan. *Compiling Nested Loop Programs to Process Networks*. PhD thesis, Universiteit Leiden, 2007. Cited in section 2.3.

[55] Nicolas Ventroux and Tanguy Sassolas. A new parallel systemc kernel leveraging manycore architectures. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2016*, pages 487–492. IEEE, 2016. Cited in sections 2.5.2 and A.16.

[56] Sven Verdoolaege. *Polyhedral Process Networks*, pages 931–965. Handbook of Signal Processing Systems. 2010. Cited in section 2.1.

[57] Samuel Williams, Andrew Waterman, and David Patterson. Roofline: an insightful visual performance model for multicore architectures. *Communications of the ACM*, 52(4):65–76, 2009. Cited in sections 2.3 and 2.4.

[58] Paul Wilmott. *Quantitative Finance*. 2006. Cited in section 1.5.

A Appendix: Details on Positioning and Added Value

The current project proposal intends to make progress in the quest for performance of dataflow programs on FPGAs, building upon the expertise of its members on algorithm design and formal methods.

The research teams that work in these domains in the Inria ecosystem are thus coming from two "themes" inside the "Algorithms, programs, software and architecture" field:

- Architecture, languages, compilation: especially CAMUS, CORSE, CAIRN and PACAP.
- Embedded systems and real time: especially SPADES and PARKAS.

Some other Inria teams belonging to other themes have also some shared research themes: ANTIQUE, CELTIQUE, and SOCRATE. There are also national non-Inria teams whose topics are close to ours.

A.1 Other teams inside LIP

The main topics of CASH (compilation, abstract interpretation, high-level synthesis) are not in the scope of any other team within the LIP laboratory.

Teams with the strongest relationships with us are:

- AVALON: The AVALON team also targets HPC applications, with a focus on the design of programming models supporting many kinds of architectures. "The team focuses in particular in energy and data intensive application profiling and modeling, data management, component based application description, and application mapping and scheduling". AVALON's activities are complementary to us since we specifically target HPC kernels (or programs designed around HPC kernels) for which we want to aggressively compute/compile an optimized (software of hardware) version. Clearly, all applications cannot be statically scheduled (or at least, only part of some large applications can be statically scheduled), and we would need efficient algorithms for our optimized kernels to be scheduled with other (possibly unknown) tasks. CASH does not target dynamic approaches like profiling that are in the scope of AVALON. AVALON considers the assembly of software components at the application level, while CASH considers computation kernels, or assembly of a few computation kernels. In other words, what CASH considers as a program is what AVALON considers as a component.
- PLUME: The activities of PLUME on program proofs focus on semantics and typing. For example, abstract interpretation is not in the scope of PLUME. PLUME works on abstract formalisms (λ-calculus, π-calculus, automata) rather than targeting concrete programming languages. We will benefit of the Plume team proximity for our activities in expressing parallel languages semantics (our vision is however more focused on efficiency than expressiveness). In the converse, PLUME's activity on program verification of parallel models could benefit from the formalization of new research problems coming from the CASH activities on optimizations.
- ROMA: Roma works on models, algorithms, and scheduling strategies at OS level. Roma's programming model considers tasks as black boxes (unlike CASH). The compilation or analyses of individual boxes are not under the scope of ROMA. However, we strongly believe that working on the scheduling of dataflow applications could lead to a better understanding of their characteristics that could benefit to ROMA in the form of more accurate task models. Also, on-the-fly compilation of tasks under resource constraint can enable new runtime scheduling strategies. These are more long term possible collaborations.

A.2 CORSE

A.2.1 Summary of CORSE's activities

CORSE's activity report is available here:

`http://raweb.inria.fr/rapportsactivite/RA2017/corse/corse.pdf`

The overall objective of CORSE[2] is to address the ever increasing complexity of applications as well as the complexity of emerging platforms by combining static and dynamic compilation techniques. The team focuses on the interaction of the program and the runtime. Its main activities are:

- The design of efficient runtimes via strong interactions with the compiler/debugger.
- Runtime verification through the generation of observers or enforcers for temporal-logic properties.
- The design of efficient compiler optimizations via the combination of static and dynamic analyses.

Only the last research direction is relevant for CASH, as we do not have any expert in debugging nor runtime. The activities of CORSE in this direction are:

- Compiler Architecture Design. The activity focuses on the design of an efficient single intermediate representation that is capable of expressing the program semantics at multiple levels (control-flow, data dependencies, profiling information). The particular focus of this activity is the design of representations that are easily maintainable and refinable at each step of static and dynamic compilation, thus can handle "information telescoping". Participants: Fabrice Rastello, Florent Bouchez Tichadou.
- Combining Static with Dynamic Analysis, Static Compiler Optimization with Run-Time Decisions. This activity aims at providing to compiler back-end the information required to perform optimizing transformations (such as combined scheduling, vectorization, register tiling, and register allocation/promotion). For that purpose, the team develops combinations of static analyses and code duplication (hybrid analyses), use intensive profiling information to gather pertinent information such as runtime dependencies, and use it to perform more accurate code optimization. Participants: Fabrice Rastello, Florent Bouchez Tichadou, Frédéric Desprez.

A.2.2 Positioning of CASH

CORSE's project intersects ours mostly with Section 2.2 for which we plan future collaborations described in Section A.2.3. There is a huge amount of work to understand irregular applications, find ways to reason on them and perform clever static and runtime optimizations, for which a single team will clearly not be sufficient.

CORSE's research on static compilation has two objectives. The first one is to estimate performance characteristics of the code (bandwidth and operational intensity estimation) in order to construct a performance model of it. The second is to perform pre-analysis that are meant to be combined with dynamic information. While we share the idea that purely static approaches will not be the definitive solutions for HPC compilation, our approach will mainly focus on the characterization of technical and scientific limits of static compilation, with a "high level language" approach.

CORSE's activity on runtime property verification is completely different from the property verification targeted by CASH. CORSE focuses on generation of observers or enforcers from possibly complex temporal logic properties, and consider programs under verification as black-boxes. As opposed to that, we analyze programs and do not consider temporal-logic properties, but only invariants.

[2] All the text concerning CORSE has been proof-read, discussed and validated by F. Rastello, head of the team.

There is a simulation activity in CORSE as well as in the current CASH proposal, but the targets are not the same: in CASH we target SoCs and FPGAs, in CORSE the objective is to generate simulators from existing architectures' ISA (instruction set).

The code generation for FPGAs is not studied in CORSE. The dynamic analyses of CORSE target classic platforms (multiprocessors, accelerators GPU/XeonPhi). CORSE does not work on abstract interpretation.

A.2.3 Possible collaborations

The activities of the "hybrid" axis of CORSE are complementary to the ones CASH plans to work on. We plan to collaborate on the following topics:

- Static analyses for dynamic compilation: CORSE's work on profiling is complementary to the fully static approach of CASH. Laure Gonnord and Fabrice Rastello already share a research project on this topic within the PROSPIEL Inria associate team:[3] the idea is to use static analyses to reduce the number of parameters to profile when optimizing an application.
- Dependencies hybrid analysis for efficient scheduling: CASH shares with CORSE the idea that the polyhedral model is not sufficient to express all data dependencies in HPC programs. Our theoretical work on dependencies will most probably have an echo in the hybrid compilation part of the CORSE's project. Indeed, we will have to deal with over-approximations of these dependencies, and computing preconditions for a code to be optimized efficiently would clearly be a way to limit the impact of these over-approximations. In the other direction, code profiling and monitoring can give hints to static code optimizers. Fabrice Rastello is a collaborator in an ANR JCJC proposal led by Laure Gonnord in Spring 2017[4], where these topics are developed in the context of data structure optimization.
- Evaluation and debugging: CORSE's activities on performance evaluation and debugging can be applied to SystemC simulations. There have already been attempts to work together on the subject via a submitted, but not funded, European project (SIM-RIDER, on Parallel and Distributed Simulation Engines) including Matthieu Moy and the CORSE team (J.-F. Méhaut).

A.3 SOCRATE

A.3.1 Summary of SOCRATE's activities

SOCRATE's activity report is available here:

`http://raweb.inria.fr/rapportsactivite/RA2016/socrate/socrate.pdf`.

The SOCRATE team is composed of 3 research directions: "Flexible Radio Front-End", "Multi-User Communications" and "Software Radio Programming Model" (also referred to as the "embedded" axis). Only the last one is relevant in a comparison with CASH (others are strongly oriented towards signal processing and information theory, and will be the subject of a new separate team called MARACAS in the near future).

The activities of the "embedded" axis of SOCRATE are: [5]

[3] The two other senior participants of this project are Sylvain Collange, from the PACAP Inria team (Rennes) and Fernando Pereira, from the Compilation Lab of the University of Minas Gerais, Brazil. The objective of this project is to develop compilation techniques that let developers code high-performance programs in high-level parallel languages such as OpenCL or NVIDIA C for CUDA, while taking maximum benefit from modern hardware.

[4] Other collaborators are Tomofumi Yuki, Inria Rennes; Carsten Fuhs, Univ Birbeck, UK; Lionel Morel, Insa Lyon, and Christophe Alias of CASH.

[5] All the text concerning SOCRATE has been proof-read, discussed and validated by T. Risset, head of the team.

- Low-power systems: impact of the capabilities of modern hardware on low-level software. A topic of growing interest is non-volatile RAM (NVRAM), which do not lose their content when powered off. This allows in particular the design of transiently powered systems able to snapshot relevant parts of their state before being powered off. NVRAM raises new challenges in memory management and in the design of operating systems. This topic is of growing importance in SOCRATE. It is supported among others by IPL ZEP[6] and an ARC6 regional PhD grant. Participants are: Kevin Marquet, Tanguy Risset, Guillaume Salagnac.
- Data-flow programming model: SOCRATE works on code generation flows using data-flow languages as input and on the monitoring of these programs. Work on code generation consider dataflow programs as a set of tasks that are mapped and scheduled on a parallel and possibly heterogeneous architecture (e.g. M. Dardaillon's PhD [20] targeting the Magali many-core processor). A large part of SOCRATE's work on data-flow consider dynamic aspects like performance monitoring, for example the numap [51] library used to identify the performance bottlenecks due to memory accesses at runtime. This activity's importance is decreasing in SOCRATE because Tanguy Risset and Kevin Marquet are strongly involved in the "low-power systems" activity, Lionel Morel is currently looking for a mobility, and the collaboration with CEA on Magali is over. SOCRATE is still interested in data-flow programming models, but currently has other priorities and would welcome another team working on the subject in Lyon. More details in section 6.3.1 of the SOCRATE activity report. Participants: Kevin Marquet, Lionel Morel, Tanguy Risset.
- Tools for FPGA development. SOCRATE's activities on FPGA development are centered around the design of efficient and precise arithmetic operators. A great success of SOCRATE is the tool FloPoCo, `http://flopoco.gforge.inria.fr/`, which allows generating arithmetic operators from a set of parameters (like precision in number of bits). SOCRATE also works on the design of numerical filters outside FloPoCo, and the integration of computing cores generated by FloPoCo in a High-Level Synthesis (HLS) flow. More details in sections 6.3.2 to 6.3.4 of the activity report. Participant: Florent de Dinechin.

A.3.2 Positioning of CASH

We share with the SOCRATE team the diagnostic that dataflow applications are an essential application domain for HPC techniques, and the fact that memory transfer issues are crucial.

However, we study dataflow applications with completely different approaches and different applications in mind. SOCRATE's work on dataflow applications focuses on dynamic monitoring and task mapping on possibly heterogeneous architectures. SOCRATE has no activity in program analysis, neither for formal verification nor for optimizing compilers. There is no current work on languages or intermediate representations in SOCRATE.

Another point where SOCRATE and CASH can be compared is hardware design. SOCRATE works on the design of efficient and precise arithmetic operators, targeting FPGA. This work targets specific operators and unlike the HLS approach followed in CASH, does not attempt to synthesize hardware circuits for general programs. Work was started to use the data-flow language GNU Radio to assemble hardware components (Internship report of Louis Besème in 2015 [11]), but in this case GNU Radio is used as an architecture description language and not as a fully-fledged programming language as it is the case with HLS approaches.

Obviously, SOCRATE and CASH target different application domains. The main target of the "embed-

[6] Inria Project Lab "Zero-power computation", `https://project.inria.fr/iplzep/teams/`

ded" axis of SOCRATE is embedded system, with an emphasis on low or ultra-low power systems. CASH targets primarily HPC applications. This has a strong impact on the scientific approach: for example memory-management techniques for transiently-powered systems are out of scope for CASH. The focus of optimization is also different: for example large matrix manipulation is a common operation in HPC, which motivates the importance of the polyhedral optimizations studied in CASH.

A.3.3 Possible collaborations

The activities of the "embedded" axis of SOCRATE are complementary to the one CASH plans to work on. The geographical proximity will allow collaboration on several topics such as:

- Data-flow programming models: SOCRATE's work on dynamic profiling of data-flow application is complementary to the static approach of CASH. Laure Gonnord already has close interactions with Lionel Morel [7] on this topic (in 2017 and 2018 they have co-supervised two students; they also co-advise a PhD starting in September 2018) and we plan to continue the collaboration which, for the moment, mainly focuses on the extension of expressivity of dataflow languages (SigmaC, Lustre) to target more code optimizations.
- Efficient hardware generation flow: CASH's work on HLS uses imperative code as input, and takes care of splitting the code into a network of communicating processes. It needs hardware implementations of arithmetic operators, and could use the work of Florent Dinechin for that. In other words, CASH works on the compiler, and SOCRATE works on the basic operators used as library by the code generated by the compiler. SOCRATE would be very interested in HLS tools for FPGA to apply them to radio applications, in particular for the CorteXlab platform: the platform includes FPGA, but radio applications are too complex to be written by hand in VHDL. SOCRATE could therefore be both a partner of CASH to design HLS tools, and a user of these tools.
- Hardware/software system simulation: SOCRATE works at the interface of software and hardware, and need simulation to prototype new solutions that can later be integrated in actual hardware. Matthieu Moy already co-supervises a PhD with SOCRATE and plans to continue the interactions with the team. SOCRATE is a user of simulation tools, while CASH will work on improving the simulation tools themselves.

In all cases, we envision mutually beneficial collaboration, but the border-line between the teams is clear, and there is very little overlap between the topics of our activities.

A.4 Synchrone (Verimag)

A.4.1 Summary of Synchrone's activities

The Synchrone team was created around the Lustre synchronous data-flow programming language[8]. The team now works on several topics, which can be found on the following web page:

`http://www-verimag.imag.fr/Synchrone-main.html`

The ones relevant in a comparison with CASH are:

- Languages and Tools for Critical Real-Time Systems. Synchrone works in particular on the Lustre language. The current main topics of interest are the implementation of critical applications on many-core architectures, and the analysis of worst-case execution time (WCET).

[7] from sept 2017, Lionel has a temporary research position at CEA LETI

[8] All the text concerning Synchrone has been proof-read, discussed and validated by P. Raymond, head of the team.

- Virtual Prototyping and Simulation. This research is done in collaboration with STMicroelectronics, and is centered on the SystemC/TLM technology for System-on-a-Chip modeling and simulation. Recent work on the topic include modeling of extra-functional properties like power-consumption and temperature and parallel execution of simulation.

A.4.2 Positioning of CASH

The Synchrone team of Verimag is currently led by Pascal Raymond, and was led by Matthieu Moy until September 2017. Activities linked to hardware simulation moved to the CASH team when Matthieu Moy joined LIP.

The common point with the Synchrone team at Verimag is the belief that the dataflow model fits particularly well the notion of task parallelism. We aim to work on the expressiveness of such languages, but we desire to go beyond the synchronous paradigm. Similarly, we share the idea that working on the language and the appropriate intermediate representations will facilitate further analyses and optimizations.

Different preferred applications (critical embedded programs for Synchrone, HPC for CASH) entail different constraints: contrarily to the Synchrone team, our objective is not to optimize the worst-case execution time (WCET), but the average performance. We will not compute WCETs for critical software. The Synchrone team does not study optimizing compilers, and targets simple compiler implementation flows to keep traceability between the source code and the generated code.

A.4.3 Possible collaborations

The presence of a former member of Synchrone within CASH will obviously create a favorable context for collaboration. However, the goal of CASH is clearly not to reproduce the Synchrone team and both teams have different objectives. We may collaborate on code generation for many-core architectures and simulation.

A.5 SPADES

We share with SPADES (`https://team.inria.fr/spades/`) the idea that the development as well as the efficient and safe compilation of programs relies on the use of adapted programming paradigms, such as the dataflow formalism. Their work on distributed synchronous programs is also a common point with us.

Like for the Synchrone team, the application domain of SPADES is embedded systems, that have specific constraints that are different from the constraints of classic HPC. We will work neither on fault tolerance, nor on the theoretical aspect of component-based programming. SPADES does not tackle the problem of optimizing compilation for HPC applications, nor circuit synthesis, nor the design of dedicated static analyses.

A.6 TEA

The Inria/Irisa TEA team (`https://team.inria.fr/tea/`) doesn't focus on HPC applications or the optimization of general purpose programs, but on the design of *correct by construction* embedded systems. Some of the synchronous activities of the team are nevertheless linked to our activities related to the design and semantics of dataflow programs or intermediate representations.

Contrarily to the TEA team, we will not focus on the modeling of time and its relative non-functional properties, nor controller synthesis.

A.7 PARKAS

We share with the PARKAS team (`http://parkas.di.ens.fr/`) the vision of a dataflow formalism for parallel programming and for the construction of compilers. Nevertheless, Marc Pouzet focuses on the resolution of front-end programming languages issues (typing, functional and higher order extensions, hybrid control and reactive systems) than on the generation of optimized code. The activities of Albert Cohen on deterministic parallel programming and the optimization of HPC kernels are complementary to ours, such as dynamic aspects in task parallelism and cross-cutting polyhedral building blocks (scheduling, code generation, embedding into a back-end compiler).

PARKAS has no activity on the design of static analyses by abstract interpretation, and does not study HLS.

A.8 Scale

A.8.1 Summary of Scale's activities

The main focus of the Scale team (`https://scale-project.github.io/`) is the safe design of large-scale distributed middleware and applications. The scientific objectives of the Scale team are focused on programming languages and distributed systems, with contributions on the semantics of distributed languages, verification of distributed applications and systems, and platforms for efficient execution of distributed applications. The Scale team developments include a Java library for distributed programming based on active objects, called ProActive[9]; ProActive is also developped by the Activeeon company[10].

A.8.2 Positioning of CASH

Ludovic Henrio is currently leading the Scale team and requested to be transferred to the LIP laboratory, and would be part of the CASH team if the transfer is accepted. The application domain of the two teams is quite distinct, as Scale is mostly interested in the optimization of distributed systems and does not focus on the efficient local execution of programs. On the contrary CASH provides optimizing compilation for local efficiency. However, the approach of the two teams relies on the design of effective programming abstractions that are both safe and easy to use, and both teams have some interest in static verification techniques. In other words, the two teams apply a quite similar approach but with very distinct application domains and scientific objectives. Also the similarity between the programming models studied by the two teams must be underlined: dataflow languages and active objects rely on the existence of separated computing entities communicating by dedicated interaction patterns. There are some dataflow aspects inherent to the active object model, and the separation of computing entities in dataflow languages is comparable to the active object language design.

A.8.3 Possible collaborations

The similarity of the approach and the studied languages open numerous possibilities for collaborations. However the inherent differences between the final objectives, e.g., optimizing compilation vs. efficient execution, local efficiency vs. large-scale distribution, static optimization vs. correctness verification, will only allow collaborations on specific subjects. From a concrete point of view two common research directions can be identified. On the practical side, the use of ProActive to coordinate programs compiled by the CASH team would be an efficient way to combine local efficient parallelism and distribution. A

[9] `https://scale-project.github.io/Software.html`
[10] `http://www.activeeon.com/`

good setting for this work is the work of Françoise Baude on scalable and adaptable stream processing applications. From a theoretical perspective, interaction with Etienne Lozes on separation logic could bring us new verification techniques for dataflow programs. In particular preliminary works of Etienne Lozes and Ludovic Henrio on separation logic for parallel methods and futures could be re-used. In both directions the relative similarity between the two programming models underlined above should play a crucial role in making the integration and the joint work easier.

A.9 Kairos

A.9.1 Summary of Kairos's activities

The Kairos team (`https://team.inria.fr/kairos/`) works on model-based design for cyber-physical systems. Their contribution range over specification, design, analysis, and verification of cyber-physical systems mostly based on formal methods and modelling with a strong focus on (logical) time.

A.9.2 Positioning of CASH

We share with Kairos the idea that formal methods can benefit concrete implementation flows, but target different systems: we target HPC where time is seen mainly as a performance constraint, while Kairos considers time as part of the specification and is interested in cyber-physical systems and internet of things. One common activity between Kairos and CASH is the simulation of heterogeneous systems.

A.9.3 Possible collaborations

We already collaborated on the subject as part of the HeLP ANR project in the past (2009-2013), where we proposed approaches to model and simulate the power consumption and temperature of Systems-on-Chips. We hope to collaborate on the subject again in the future. Additionally, Ludovic Henrio has already collaborated with members of Kairos both in the context of exchanges with ECNU Shanghai and on the topic of behavioral specification and verification of distributed applications (more than 20 joint publications). Some joint work on this last topic is still ongoing in collaboration with Rabéa Ameur Boulifa (Eurecom, Télécom ParisTech).

A.10 PACSS (Verimag), ANTIQUE (Paris), CELTIQUE (Rennes)

A.10.1 Summary of PACSS activities

PACSS (`http://www-verimag.imag.fr/PACSS.html?lang=fr`) works on program analysis and verification[11]. It follows several directions:

- Static analysis (in particular Abstract Interpretation) to prove properties or extract invariants from programs.
- Static analysis applied to security.
- Interactive proofs (especially using the Coq proof assistant).

Only the first axis is relevant in a comparison with CASH. Its main participants are David Monniaux and Nicolas Halbwachs.

[11] All the text concerning PACSS has been proof-read, discussed and validated by D. Monniaux, head of the team.

A.10.2 Summary of ANTIQUE activities

The ANTIQUE team (`https://www.di.ens.fr/AntiqueTeam.html.fr`) studies:[12]

- abstraction techniques in general (abstract interpretation of course, but also modeling techniques in biology);
- abstract domains and static analysis techniques
- applications to software verification with a wide spectrum of software kinds (synchronous, parallel, concurrent, data-structure intensive, numeric...) and properties (safety, security, liveness).
- applications to other fields, such as biology.

Only the activities concerning static analyses of general purpose programs (or safety-critical ones, with the Astree Analyzer [12]) are relevant for a comparison with CASH. The participants are Xavier Rival, Cezara Dragoi, Jerôme Féret and Vincent Danos.

A.10.3 Summary of CELTIQUE activities

CELTIQUE (`https://www.irisa.fr/celtique/`) works on programs reliability and security[13], especially in the following directions:

- Program analyses by abstract interpretation, and decision procedures.
- Formal certification of static analysis tools and compilers.
- Application to software security.

The activities on abstract interpretation and on formal semantics of intermediate representations inside compilers (SSA), are relevant for a comparison with CASH. The participants are Delphine Demange, David Pichardie, Sandrine Blazy, Fréderic Besson, and Thomas Jensen.

A.10.4 Positioning of CASH

These three teams have activities on abstract interpretation that meet ours. We share the ambition of developing expressive and usable analyzers, that scale.

Nevertheless, the work on suitable intermediate representations and the design of static analyses for optimizing compilers is only common with some researchers of the CELTIQUE team (Delphine Demange, David Cachera). The constraints of an analysis for safety or compilation are different: the former favor precision and the second has higher performance constraints. CELTIQUE has complementary activities on compilation: they share our credo that compilers should be built on solid semantic foundations, as shown by their involvement in the CompCert project, but they do not specifically develop new compiler optimizations.

Furthermore, PACSS, ANTIQUE and CELTIQUE are only focusing on software while we also target hardware verification and synthesis.

A.10.5 Possible collaborations

Both David Monniaux and Nicolas Halbwachs are former members of the Synchrone team led by Matthieu Moy up to 2017. PACSS and Synchrone still have very close interactions. David Monniaux and Matthieu Moy already co-supervised 3 internships and 1 PhD. David Monniaux and Laure Gonnord already co-authored three conference papers. The context is therefore very favorable to collaboration:

[12] All the text concerning ANTIQUE has been proof-read, discussed and validated by X.Rival, head of the team.
[13] All the text concerning CELTIQUE has been proof-read, discussed and validated by T. Jensen, head of the team.

- Laure Gonnord and David Monniaux have an ongoing work on designing array abstractions, and a first paper in SAS ([39]) which is strongly related on the fundamental research on computation dependencies of the CASH project.
- Since January 2017, we organize a common weekly "Reading Group", via teleconference, on the static analysis topic[14].

We do not currently have any projects with ANTIQUE, but we could, for example on topics related to the Anastasec ANR project[15]. The goal of this project is the automatic proof of security properties on low level codes, which could be instantiated in our particular context of HLS, and for which a deep work on scalability is imperative.

We do not share any projects with CELTIQUE. With Delphine Demange and David Pichardie we could collaborate on developing certified compiler optimizations, as a medium-term project.

A.11 CAIRN

A.11.1 Summary of CAIRN's activities

CAIRN focuses on new architectures, algorithms, and design methods to design flexible and energy efficient hardware. Its main activities are:

- The design of new reconfigurable architectures with an emphasis on flexible arithmetic, dynamic reconfiguration management and low power consumption.
- The design of dedicated synthesis algorithms and compiler optimizations.
- Applications to wireless networks and cryptography.

Only the second axis is common with CASH. On this axis, CAIRN has a large spectrum of activities: source-level polyhedral transformations for HLS, compiler support for cycle-level runtime reconfiguration, runtime memory disambiguation or models for stencil compilation on FPGA to quote a few.

A.11.2 Positioning of CASH

We share the same polyhedral culture as CAIRN. Consequently, we deal with the same problematics: How to scale the polyhedral analysis? How to reduce the complexity of the code generated? Which non-polyhedral applications can be handled? However, CASH believes that dataflow models is a central object in the design of compiler-assisted parallelization, in particular for circuit synthesis. Hence, we will rather focus on compiler analysis for dataflow models. As mentioned above, CAIRN has a large spectrum of activities around energy efficient reconfigurable computing, including the design of reconfigurable architectures, compiler support and applications. In contrast, the perimeter of CASH is centered around the development of parallel dataflow languages and their compilers.

A.11.3 Possible collaborations

The activities of the "compiler optimization" direction of CAIRN tackle the same issues as CASH with a complementary approach. We believe that a collaboration is possible on the runtime machinery required to compile irregular applications with polyhedral techniques.

[14] `http://stator.imag.fr/w/index.php/VerifGroup`
[15] `https://www.di.ens.fr/~feret/anastasec/`

A.12 MOCS

MOCS focuses on methods for modeling, estimation and optimization of heterogeneous multiprocessor system-on-chip (MPSoC) including application-specific dedicated architectures, reconfigurable/self-adaptive architectures as well as coarse-grain reconfigurable architectures. MOCS aims at defining a design continuum between applications embedded software, CAD tools and hardware architectures. Its main activities are:

- High-level synthesis
- Architectures for ultra-low power processing
- Model-driven approaches and model checking
- Memory-based computing
- Hardware-level attack detection

The HLS algorithms developed at MOCS leverage the standard intermediate representations for HLS (hierarchical control/data flow graphs) developed in the HLS tool of the team, GAUT. CASH will leverage a polyhedral intermediate representation, which should lead to different approaches to address the same issues (parallelism, resource, throughput). This could be a point of convergence with CASH.

A.13 DATAMOVE

Datamove shares the same application domain (HPC) as CASH, and also deals with data-management in parallel applications, but at a different scale. CASH studies optimizations at the scale of one machine, for example optimize the arithmetic intensity to make as many accesses to the cache and avoid accessing the main memory. Datamove, on the other hand, studies data movement at the scale of a supercomputer or datacenter. The solutions are therefore different: heavyweight dynamic algorithms can be used at large scale, while we target algorithms with low or no dynamic overhead, hence focus on static optimization. Datamove does not study compilation. Datamove recently developed the FlowVR framework that allows coarse-grained dataflow programming. This could be a point of convergence with CASH.

A.14 Other Inria teams of the theme "Architecture, languages, compilation"

A.14.1 CAMUS

The CAMUS team targets the conception of a production chain for efficient execution of an application. CAMUS has an activity on static parallelization and optimization, which is complementary with dynamic parallelization via profiling and a work on speculative models. We will not study dynamic compilation and speculative models, nor runtime systems. Rather, we will focus on the static part of compilation and synthesis.

CAMUS has also an activity on the formal proof of transformations, that we will not work on. CAMUS focuses on multicore architectures, and does not target HLS.

A.14.2 PACAP

The PACAP team works on two topics related to ours: timing analysis for embedded systems (Worst-case Execution Time, WCET, analysis and scheduling) and dynamic compilation. We plan to use WCET-related techniques in our analysis (for load balancing for example), but unlike PACAP we do not target embedded or critical systems. Regarding the compilation activities, the fundamental distinction is the focus on static techniques in CASH.

A.15 System-Level Synthesis (TIMA)

The SLS team of TIMA is known for its expertise in simulation, in particular using SystemC/TLM. The SLS team focuses on techniques to integrate embedded software in the platform: dynamic binary translation and native simulation. CASH does not plan to work on these subjects. Also, SLS targets simulations with precise timing, while CASH plans to work on loosely timed simulations, which raise completely different issues especially when it comes to parallelization [10].

SLS also focuses on high-level synthesis for HPC and embedded systems targeting FPGA. Their tool, AUGH, relies on design-space exploration associated with fast hardware synthesis [47]. CASH targets program optimizations based on the polyhedral model, which are more accurate, but more expensive; hence less adapted to design space exploration. Thus our approaches are complementary. Also, SLS focuses on HLS-level system mechanisms for FPGAs. Recently, they developed a context switch on hardware tasks executed on reconfigurable systems [15]. CASH does not plan to work on these subjects. Our purpose is to focus on resource efficient synthesis of compute-intensive kernels.

A.16 Département Architectures Conception et Logiciels Embarqués (CEA LIST)

The "Département Architectures Conception et Logiciels Embarqués" (DACLE) department of CEA-LIST works on simulation with SystemC/TLM, with an emphasis on many-core processor, and parallel/heterogeneous simulation. Tanguy Sassolas and Nicolas Ventroux already proposed a parallel simulation solution called SCale [55].

One particularity of the department is the proximity between computer-science researchers and hardware designers: simulation tools are used internally at CEA to develop new hardware platforms.

We started collaborating with CEA-LIST in 2016 through a European project submission (which was unfortunately not funded). We started co-supervising the PhD of Gabriel Busnot on parallel and heterogeneous simulation. There are several directions to explore (a PhD will likely be too short to explore them all), like dealing with non-trivial memory management, cosimulation with dedicated physics solvers, and dealing with various levels of precision in a single platform.

A.17 CIAN - Analog and Digital Integrated Circuit (LIP6)

The CIAN team of LIP6 works on design methods for analog and digital components integrated on the same chip. This includes simulation, and the team has been working on topics close to our simulation-related activities: a parallel execution system for SystemC/TLM called TLM-DT [38], and a simulator able to mix execution models called SystemC-MDVP [8].

TLM-DT has an important drawback: it requires rewriting substantial parts of the platform in a different modeling style to get efficient and correct simulation. It is however a good source of inspiration that we already used in our work. SystemC-MDVP is a very good starting point to build heterogeneous co-simulations. One missing point however is that SystemC-MDVP does not support parallel execution yet.

A.18 International projects and teams

A.18.1 High-Level Synthesis (HLS)

- The **VAST** laboratory (VLSI Architecture, Synthesis, and Technology) led by Prof. Jason Cong from University of California Los Angeles targets customized computing for big data applications, energy-efficient computing and electronic design automation. VAST has many achievements, including source-level optimizations for FPGA: data transfer optimization, parallelization or multibank mem-

ory allocation to quote a few. CASH shares the same goals as VAST, but with a slightly different approach. Our goal is rather to design dataflow intermediate representation tailored to the requirements of reconfigurable computing with the corresponding hardware generation algorithms. We believe that a collaboration is possible either on source-level polyhedral analysis or on low-level hardware generation.

- **ROCCC** (Riverside optimizing compiler for configurable computing) is an HLS project led by Prof. Walid Najjar from University of California Riverside, USA. It features compiler optimizations as parallelism extraction, data reuse or design space exploration. Though the project is more oriented to embedded systems, many ideas could profit to HPC, as their hardware mechanism to fetch data for sliding window loop nest (smart buffer) or to optimize irregular data accesses. We believe that a collaboration is possible on the hardware generation part.
- The **Compaan** project was initially led by Prof. Ed Depreterre, Bart Kienhuis, and Todor Stefanov from University of Leiden, Netherlands. It aims at providing an HLS tool for accelerating compute-intensive embedded applications. The Compaan project came up with the interesting concept of Polyhedral process network (PPN), a dataflow representation tailored to regular kernels. CASH shares the idea that dataflow representations are fundamental to design reusable HLS analysis and optimizations. However, the PPN model is too restricted for our purpose. We believe that a fruitful collaboration is possible on the dataflow model part.
- The **Circuits and Systems group** led by George A Constantinides from Imperial College, London, aims at studying architectures, synthesis tools, and applications of customized hardware. Their achievements range from hardware mechanisms to HLS optimizations. For example, they proposed a source-level data transfer optimization. They also investigate HLS for irregular applications. We think a collaboration is possible for most of the issues we plan to investigate.
- The **Panda** project led by Fabrizio Ferrandi from Politecnico di Milano (Italy) investigates HLS specific issues concerned with parallelism extraction, hardware/software partitioning and mapping, metrics for performance estimation of embedded software applications and dynamic reconfigurable devices. CASH does not target embedded systems. However, we believe that a collaboration is possible on source-level analysis and optimizations.

A.18.2 Simulation of Systems-on-a-Chip

- The **group of computer architecture at university of Bremen** (head: Rolf Drechsler) works among other topics on simulation, in particular SystemC. Initially, the team worked essentially on low levels of abstraction like gate-level or Register Transfer Level (RTL). Activities with high-level models like TLM (which is the scope of CASH) target tools to help the programmer understand a program (fault localization, visualization) and program verification. The first topic is not in the scope of CASH, but we may collaborate on the second.
- The **Institute for Communication Technologies and Embedded Systems (ICE), Aachen University** (head: Rainer Leupers) has activities very close to the simulation axis of CASH: they also work on parallel execution of SystemC/TLM programs. However, their work target models with fine-grained timing, and we showed that this category of approach is not applicable to the models we target in CASH [10].
- The **Center for Embedded and Cyber-physical Systems (CECS), University of California**, in particular Rainer Dömer, work on parallel simulation of hardware (originally in SpecC, but also on SystemC). The approach followed by CECS is very different from ours, as it targets lower-level simulations like RTL and cycle-accurate models. On these models, a high degree of parallelism exists at a

very fine granularity. One of the challenges successfully tackled by CECS is the dependency analysis to allow out-of-order parallel simulation while preserving the semantics of the model [17].

A.18.3 Programming Languages and Compilation

- **University of Uppsala – Programming language research group**. Several research groups in Europe work on actors and similar languages with different objectives. Among them, the Programming language research group in Uppsala designs and implements the Encore programming model aiming at efficient execution of active objects. This research group and other partners of the Upscale European project target the efficient execution of active objects by compilation into C, and execution using the PonyRT actor runtime environment for C. They also use advanced typing techniques to optimize the execution. The languages we target are relatively different from Encore and the semantics we will ensure will be much more deterministic, we also aim at lower-level optimization than the object allocation of Encore but some collaborations would be useful, especially in the context of language design.

- The **Compiler Technology and Parallel Computing at PELAB** (U. Linköping) works on a hybrid library/compilation based approach called SkePU. Users write code using a library providing several programming "skeletons" (e.g. Map, Reduce, ...), and a source-to-source transformation with multiple backends generates code for efficient execution on CPUs, GPUs or others. SkePU allows the user to express parallelism in a flexible way, and provides optimized runtimes to run the generated executable. As opposed to CASH, SkePU does not do parallelism extraction nor sophisticated code analysis.

- The **Software Performance Optimization Research Group (London)** (head: Paul Kelly), studies compiler technology, with a specific emphasis on domain-specific (scientific applications) performance optimizations. Paul Kelly was one of Laure Gonnord's habilitation reviewer. The kind of analyses that are studied by the team are clearly more specific that ours, however we could in a middle-term collaborate on specific topics like kernel optimizations for sparse matrices or meta-programming for loop optimization.

- The **Compiler Lab at UFMG (Brasil)** (head : F. M. Q. Pereira) aims at designing new compiler optimizations for general purpose compilers, like LLVM. Their activities are closely linked to our axis of research developed in Section 2.2, for which we have a long term collaboration.

- The **Scalable Parallel Computing Laboratory, in Zurich (Switzerland)** (head: Torsten Hoefler) focuses on all kinds of optimizations for large-scale parallel applications. The group studies static as well as dynamic analyses and optimizations. We have a starting collaboration with Tobias Grosser, postdoc, to design new optimizations based on the polyhedral model inside the LLVM compiler (related to Section 2.2).

- The **Mélange group in Colorado State University** (head: Sanjay Rajopadhye) studies programming languages, algorithms, compilers, computer architecture, VLSI algorithms, embedded systems and FPGAs in the light of the polyhedral model. We already co-advised a PhD thesis and maintain an informal collaboration with Prof. Sanjay Rajopadhye, the team leader. This activity is related to Section 2.3 and Section 2.4.

A.18.4 Static analysis and verification groups

The following teams work among other topics on software and hardware verification:

- The **Automatic Verification Research Group, at University of Oxford (UK)**, in particular Daniel Kroening, works on various topics linked with formal verification and correct-by-construction synthesis for software and hardware systems. The research led by this group, in particular in the "Hardware verification" and "Model-checking" target real-life applications. Some of the major achievements of the group are the symbolic model-checkers PRISM and FPRs, whose expressiveness and scalability have been successfully shown on real-case industrial studies. As for hardware verification, we could collaborate especially in the context of checking consistency of our hardware generation process.
- The **Programming Languages and Systems Research Group, at University of Kent (UK)** (head: Andy King) focuses on the practical and theoretical aspects of language design. The activities of the group are diverse from type systems and semantics to runtime verification. We could be interested in collaborating on various aspects of these research activities, like abstract interpretation with Andy King or semantics of parallel programs with Stefan Maar.
- The **Software Engineering Group, at University of Freiburg (Germany)** (head: Andreas Podelski) works on software verification. Andreas Podelski was one of Laure Gonnord's habilitation reviewer. The group focuses on developing semantic-based verification techniques based on various abstractions, and develop tools that delegate part of their work to dedicated solvers (constraint programming, decision procedures) for which they have also major contributions. The analyses tackled by this group are too costly for our specific compilation analyses, however we could collaborate on specific topics like the use of specific solvers in order to improve the expressiveness of our analyses.
- The **Forsyte Group, at TU Wien (Austria)**, in particular Laura Kovács, conduct research activities in the area of formal methods for critical software systems, to prove and infer functional as well as non functional properties. Their particular expertise is numerical invariant generation, and also the design of methods and tools to infer bounds on memory consumptions. Like the preceding group, the analyses developed are way too costly for our usage, however, the formalization of memory consumptions problems may be a source of inspiration.
- The **IMDEA Software lab, in Madrid (Spain)** (head: Manuel Carro), in particular Manuel Hermenegildo and John Gallagher, studies various subjects in the domain of software development, from language design to advanced expressive verification tools. Among the three research directions, the *Program Analysis and Verification* and *Languages and Compilers* are the closest to ours. Their particular expertise is on the design of (expensive) logic-based program verification processes, but they also have conducted research in developing models for cache and memory performance, which might inspire us.
- The **Spy Lab at University of Verone (Italy)**, especially the group lead by Roberto Giacobazzi, designs formal methods and new abstract domains with application to program analysis. Although they focus more on security and malware detection, their work on the modeling of obfuscation algorithms can be an inspiration for us for semantic-based compiler optimizations.
- Other static analyses contributors in Italy include Enea Zaffanella in Parma and Francesca Scozzari in Pescara. They both design numerical abstract domain with a particular focus on the scalability of the polyhedra abstract domain.
- The **Formal Methods - for Safe and Secure Systems section, at the T.U. of Danemark** (head: Hanne Riis Nielson) develop methods and tools to facilitate the development of safe and secure IT sys-

tems with good performance. They have activities in logics and semantics for a large variety of applications (web systems, cloud, business process).

- The **Secure, Reliable, and Intelligent Systems Lab at ETH Zurich (Switzerland)** (head: Martin Vechev) works on interdisciplinary approaches to design reliable programs and systems. The activity which is the most related to ours is the one linked to the design of fast numerical abstract domains.
- The **Lab for Automated Reasoning and Analysis at EPFL (Lausanne, Switzerland)** (head: Viktor Kuncak) also aims at analyzing general purpose programs. They design solutions that involve queries to logic and constraint solvers, like SMT-solvers.
- In the US, the main contributors to the static analysis research area are F. Logozzo at **Facebook**, Patrick Cousot at **New York University**, and Thomas Reps at the **University of Wisconsin**. The main focus of these activities are the design of expressive abstract domains for safety analysis. Thomas Reps has various contributions on designing memory and numerical analysis, and now is one of the members of the Pliny project (`http://pliny.rice.edu/index.html`, leader Vivek Sarkar) whose objective is to design scalable tools to analyze big chunks of code. This project is closely linked to the Section 2.2 of this proposal.