Activity Report 2018

Team CASH

Compilation and Analysis, Software and Hardware

Inria teams are typically groups of researchers working on the definition of a common project, and objectives, with the goal to arrive at the creation of a project-team. Such project-teams may include other partners (universities or research institutions).

RESEARCH CENTER
Grenoble - Rhône-Alpes

THEME
Architecture, Languages and Compilation

Table of contents

Team CASH

Creation of the Team: 2018 April 01

Keywords:

Computer Science and Digital Science:

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

Other Research Topics and Application Domains:

B9.5.1. - Computer science

1. Team, Visitors, External Collaborators

Research Scientists

Christophe Alias [Inria, Researcher]
Ludovic Henrio [CNRS, Researcher, from Oct 2018, HDR]

Faculty Members

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

PhD Students

Julien Braine [Ecole Normale Supérieure Lyon, from Sep 2018]
Paul Iannetta [Ecole Normale Supérieure Lyon, from Sep 2018]

Interns

Bilel Aouadhi [Univ de Claude Bernard, until Jul 2018]
Alexandra Dobre [Inria, from Jul 2018, until Sep 2018]
Arthur Gontier [Ecole Normale Supérieure Lyon, until Jun 2018]
Nicoleta Ligia Novacean [Inria, from Jul 2018 until Sep 2018]

Administrative Assistant

Laetitia Gauthe [Inria, until Nov 2018]

2. Overall Objectives

2.1. Introduction

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

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

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

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

These issues are considered as one of the main challenges in the Hipeac roadmap [27] which, among others, cites the two major issues:

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

2.2. Overall Objectives

The overall objective of the CASH team is to take advantage of the characteristics of the specific hardware (generic hardware, hardware accelerators, or reconfigurable chips) to *compile energy efficient software and hardware*. More precisely, 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 3.1).
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 3.2).

3. Transformations from and to the dataflow representation, combining traditional tools dedicated to dataflow and specific methods like the polyhedral model (Section 3.3).
4. A high-level synthesis (HLS) tool, built on the above item (instantiated with the particularities of FPGAs) and a code generation tool (Section 3.4). This HLS tool 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 3.5).

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

Note that these directions are strongly tied together. We use 5 research 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.

3. Research Program

3.1. Definition of dataflow representations of parallel programs

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

The main goal of the CASH team is to provide efficient analyses and the optimizing compilation frameworks for dataflow programming models. The results of the team 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 [55] 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 [37] or Streamit [51], where the scheduling is fully static, and general communicating networks like KPNs [39] or RVC-Cal [19] 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 [26] that nowadays introduce more and more dataflow primitives to enable data-driven interactions between agents, particularly with *futures* [24], [31].

3.1.1. Expected Impact

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

3.1.2. Scientific Program

3.1.2.1. Short-term and ongoing activities.

We obtained preliminary experimental [16], [17], [32] and theoretical [38] results, exploring several aspects of dataflow models. The next step is to define accurately the intermediate dataflow model and to study existing programming and execution models:

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

3.1.2.2. Medium-term activities.

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

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

3.1.2.3. Long-term activities.

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

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

3.2. Expressivity and Scalability of Static Analyses

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

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

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

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

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

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

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

Scalable abstract domains. We already have experience in designing low-cost semi relational abstract domains for pointers [44], [42], as well as tailoring static analyses for specialized applications in compilation [30], [50], Synchronous Dataflow scheduling [49], and extending the polyhedral model to irregular applications [15]. We also have experience in the design of various static verification techniques adapted to different programming paradigms.

3.2.1. Expected impact

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

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

3.2.2. Scientific Program

3.2.2.1. Short-term and ongoing activities.

Together with Paul Iannetta and Lionel Morel (INSA/CEA LETI), we are currently working on the *semantic rephrasing* of the polyhedral model [34]. The objective is to clearly redefine the key notions of the polyhedral model on general flowchart programs operating on arrays, lists and trees. We reformulate the algorithms that are performed to compute dependencies in a more semantic fashion, i.e. considering the program semantics instead of syntactical criteria. The next step is to express classical scheduling and code generation activities in this framework, in order to overcome the classical syntactic restrictions of the polyhedral model.

3.2.2.2. Medium-term activities.

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

At the same time, we plan to continue to work on scaling static analyses for general purpose programs, in the spirit of Maroua Maalej's PhD [41], 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 [36] that tries to formalize the cost/precision compromise of interprocedural analyses with respect to a "context sensitivity parameter".

3.2.2.3. Long-term activities.

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

An ambitious challenge is to find a generic way of adapting existing (relational) abstract domains within the Single Static Information [20] 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 [25] which gave a theoretical scheme for classical abstract interpretation analyses.

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

3.3. Compiling and Scheduling Dataflow Programs

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

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

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

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

Translating a sequential program into a medium-grain dataflow model. The programs considered are compute-intensive parts from HPC applications, typically big HPC kernels of several hundreds of lines of C code. In particular, we expect to analyze the process code (actors) from the dataflow programs. On short ACL (Affine Control Loop) programs, direct solutions exist [53] and rely directly on array dataflow analysis [29]. On bigger ACL programs, this analysis no longer scales. We plan to address this issue by *modularizing* array dataflow analysis. Indeed, by splitting the program into processes, the complexity is mechanically reduced. This is a general observation, which was exploited in the past to compute schedules [28]. 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 [56] 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) [17]. 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 [35]. In the presence of big agents, the programmer is left with the splitting (or merging) of these agents by-hand if she wants to further parallelize her program (or at least give this opportunity to the runtime, which in general only sees agents as non-malleable entities). In the presence of arrays and loop-nests, or, more generally, some kind of regularity in the agent's code, however, we believe that the programmer would benefit from automatic parallelization techniques such as those proposed in the previous paragraphs. To achieve the goal of a totally integrated approach where programmers write the applications they have in mind (application flow in agents where the agents' code express potential parallelism), and then it is up to the system (compiler, runtime) to propose adequate optimizations, we propose to build on solid formal definition of the language semantics (thus the formal specification of parallelism occurring at the agent level) to provide hierarchical solutions to the problem of compilation and scheduling of such applications.

3.3.1. Expected impact

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

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

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

3.3.2. Scientific Program

3.3.2.1. Short-term and ongoing activities.

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

Together with Lionel Morel (INSA/CEA LETI), we currently work on the evaluation of the practical advantage of combining the dataflow paradigm with the polyhedral optimization framework. We empirically build a proof-of-concept tooling approach, using existing tools on existing languages [33]. We combine dataflow programming with polyhedral compilation in order to enhance program parallelization by leveraging both inter-agent parallelism and intra-agent parallelism (i.e., regarding loop nests inside agents). We evaluate the approach practically, on benchmarks coming from image transformation or neural networks, and the first results demonstrate that there is indeed a room for further improvement.

3.3.2.2. Medium-term activities.

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

We plan to investigate in the following directions:

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

3.3.2.3. Long-term activities.

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

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

3.4. HLS-specific Dataflow Optimizations

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

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

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

Our results will be evaluated by using existing HLS tools (e.g., Intel HLS compiler, Xilinx Vivado HLS). We will implement our compiler as a source-to-source transformation in front of HLS tools. With this approach, HLS tools are considered as a "back-end black box". The CASH scheme is thus: (i) *front-end*: produce the CASH dataflow representation from the input C kernel. Then, (ii) turn this dataflow representation to a C program with pragmas for an HLS tool. This step must convey the characteristics of the dataflow representation found by step (i) (e.g. dataflow execution, fifo synchronisation, channel size). This source-to-source approach will allow us to get a full source-to-FPGA flow demonstrating the benefits of our tools while relying on existing tools for low-level optimizations. Step (i) will start from the DCC tool developed by Christophe Alias, which already produces a dataflow intermediate representation: the Data-aware Process Networks (DPN) [17]. 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 [56] 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.

3.4.1. Expected Impact

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

3.4.2. Scientific Program

3.4.2.1. Short-term and ongoing activities.

The HLS compiler designed in the CASH team currently extracts a fine-grain parallel intermediate representation (DPN [17], [16]) 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 [45].

3.4.2.2. Medium-term activities.

Our DPN-to-C code generation will need to be improved in many directions. The first point is the elimination of redundancies induced by the DPN model itself: buffers are duplicated to allow parallel reads, processes are produced from statements in the same loop, hence with the same control automaton. Also, multiplexing uses affine constraints which can be factorized [18]. We plan to study how these constructs can be factorized at C-level and to design the appropriate DPN-to-C translation algorithms.

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

3.4.2.3. Long-term activities.

The DPN-to-C compiler 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 [58]. 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.

3.5. Simulation of Hardware

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

Because of the increasing complexity of both software and hardware, traditional simulation techniques (Register Transfer Level, RTL) are too slow to allow full system simulation in reasonable time. New techniques such as Transaction Level Modeling (TLM) [14] in SystemC [13] have been introduced and widely adopted in the industry. Internally, SystemC uses discrete-event simulation, with efficient context-switch using cooperative scheduling. TLM abstracts away communication details, and allows modules to communicate using function calls. We are particularly interested in the loosely timed coding style where the timing of the platform is not modeled precisely, and which allows the fastest simulations. This allowed gaining several orders of magnitude of simulation speed. However, SystemC/TLM is also reaching its limits in terms of performance, in particular due to its lack of parallelism.

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

We 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 [23]. We focus on semi-automatic approaches where the programmer only needs to make minor modifications of programs to get significant speedups.

3.5.1. Expected Impact

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

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

3.5.2. Scientific Program

3.5.2.1. Short-term and ongoing activities.

We started the joint PhD (with Tanguy Sassolas) of Gabriel Busnot with CEA-LIST. The research targets parallelizing SystemC heterogeneous simulations. CEA-LIST already developed SCale [54], which is very efficient to simulate parallel homogeneous platforms such as multi-core chips. However, SCale cannot currently load-balance properly the computations when the platform contains different components modeled at various levels of abstraction. Also, SCale requires manual annotations to identify accesses to shared variables. These annotations are given as address ranges in the case of a shared memory. This annotation scheme does not work when the software does non-trivial memory management (virtual memory using a memory management unit, dynamic allocation), since the address ranges cannot be known statically. We started working on the "heterogeneous" aspect of simulations with an approach allowing changing the level of details in a simulation at runtime, and started tackling the virtual and dynamic memory management problem by porting Linux on our simulation platform.

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

We are working in two directions. The first is to provide user-friendly tools to allow graphical inspection of traces. For example, we 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.

3.5.2.2. Medium-term activities.

Several research teams have proposed different approaches to deal with parallelism and heterogeneity. Each approach targets a specific abstraction level and coding style. While we do not hope for a universal solution, we believe that a better coordination of different actors of the domain could lead to a better integration of solutions. We could imagine, for example, a platform with one subsystem accelerated with SCale [54] from CEA-LIST, some compute-intensive parts delegated to sc-during [43] from Matthieu Moy, and a co-simulation with external physical solvers using SystemC-MDVP [21] 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.

3.5.2.3. Long-term activities.

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

4. Application Domains

4.1. Compute-intensive Loop Kernels

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 [46]. The irregular versions can be found in [47]. Numerical kernels used in quantitative finance [57] are also good candidates, e.g., finite difference and Monte-Carlo simulation.

4.2. Medium-size HPC applications

The medium-size applications we target are streaming algorithms [19], scientific workflows [52], and also the now very rich domain of deep learning applications [40]. We explore the possibilities of writing (see Section 3.1) and compiling (see Section 3.3) applications using a dataflow language. As a first step, we will target dataflow programs written in SigmaC [22] 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.

Obviously, large applications are not limited to assembly of compute kernels. Our languages and formalism definitions (3.1) and analyses (3.2) 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 (3.5) 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.

5. Highlights of the Year

5.1. Highlights of the Year

- CASH has been validated as a *équipe projet commune* (EPC) by the *comité des projets*.
- We designed a dataflow transformation which always recovers all the FIFO in our dataflow model (DPN) after a loop tiling [1], [9], a program transformation widely used in automatic parallelization. This is an important enabling transformation which reinforces DPN as an intermediate representation in the CASH HLS project.
- We obtained new results on the comparison between different forms of synchronisation on futures, bringing a better understanding on the impact dataflow synchronisation and future typing on program synchronisation.

6. New Software and Platforms

6.1. DCC

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

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

6.2. PoCo

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

- Participant: Christophe Alias
- Contact: Christophe Alias

6.3. MPPcodegen

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

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

7. New Results

7.1. Monoparametric Tiling of Polyhedral Programs

Participant: Christophe Alias.

Tiling is a crucial program transformation, adjusting the ops-to-bytes balance of codes to improve locality. Like parallelism, it can be applied at multiple levels. Allowing tile sizes to be symbolic parameters at compile time has many benefits, including efficient autotuning, and run-time adaptability to system variations. For polyhedral programs, parametric tiling in its full generality is known to be non-linear, breaking the mathematical closure properties of the polyhedral model. Most compilation tools therefore either perform fixed size tiling, or apply parametric tiling in only the final, code generation step.

We introduce monoparametric tiling, a restricted parametric tiling transformation. We show that, despite being parametric, it retains the closure properties of the polyhedral model. We first prove that applying monoparametric partitioning (i) to a polyhedron yields a union of polyhedra, and (ii) to an affine function produces a piecewise-affine function. We then use these properties to show how to tile an entire polyhedral program. Our monoparametric tiling is general enough to handle tiles with arbitrary tile shapes that can tesselate the iteration space (e.g., hexagonal, trapezoidal, etc). This enables a wide range of polyhedral analyses and transformations to be applied.

This is a joint work with Guillaume Iooss (Inria Parkas) and Sanjay Rajopadhye (Colorado State University).

This work is under submission [12].

7.2. Improving Communication Patterns in Polyhedral Process Networks

Participant: Christophe Alias.

Process networks are a natural intermediate representation for HLS and more generally automatic parallelization. Compiler optimizations for parallelism and data locality restructure deeply the execution order of the processes, hence the read/write patterns in communication channels. This breaks most FIFO channels, which have to be implemented with addressable buffers. Expensive hardware is required to enforce synchronizations, which often results in dramatic performance loss. In this paper, we present an algorithm to partition the communications so that most FIFO channels can be recovered after a loop tiling, a key optimization for parallelism and data locality. Experimental results show a drastic improvement of FIFO detection for regular kernels at the cost of a few additional storage. As a bonus, the storage can even be reduced in some cases.

This work has been published in the HIP3ES workshop [1]

7.3. FIFO Recovery by Depth-Partitioning is Complete on Data-aware Process Networks

Participant: Christophe Alias.

In this paper, we build on our algorithm for FIFO recovery based on depth partitioning. We describe a class of process networks where the algorithm can recover all the FIFO channels. We point out the limitations of the algorithm outside of that class. Experimental results confirm the completeness of the algorithm on the class and reveal good performance outside of the class.

This work is under submission [9]

7.4. Parallel code generation of synchronous programs for a many-core architecture

Participant: Matthieu Moy.

Embedded systems tend to require more and more computational power. Many-core architectures are good candidates since they offer power and are considered more time predictable than classical multi-cores. Data-flow Synchronous languages such as Lustre or Scade are widely used for avionic critical software. Programs are described by networks of computational nodes. Implementation of such programs on a many-core architecture must ensure a bounded response time and preserve the functional behavior by taking interference into account. We consider the top-level node of a Lustre application as a software architecture description where each sub-node corresponds to a potential parallel task. Given a mapping (tasks to cores), we automatically generate code suitable for the targeted many-core architecture. This minimizes memory interferences and allows usage of a framework to compute the Worst-Case Response Time.

This is a joint work with Amaury Graillat, Pascal Raymond (IMAG) and Benoît Dupont de Dinechin (Kalray).

This work has been published at the DATE conference [6].

7.5. Estimation of the Impact of Architectural and Software Design Choices on Dynamic Allocation of Heterogeneous Memories

Participant: Matthieu Moy.

Reducing energy consumption is a key challenge to the realization of the Internet of Things. While emerging memory technologies may offer power reduction, they come with major drawbacks such as high latency or limited endurance. As a result, system designers tend to juxtapose several memory technologies on the same chip. This paper studies the interactions between dynamic memory allocation and architectural choices regarding this heterogeneity. We provide cycle accurate simulations of embedded platforms with various memory technologies and we show that different dynamic allocation strategies have a major impact on performance. We demonstrate that interesting performance gains can be achieved even for a low fraction of heap objects in fast memory, but only with a clever data placement strategy between memory banks.

This is a joint work with Tristan Delizy, Kevin Marquet, Tanguy Risset, Guillaume Salagnac (Inria Socrate) and Stéphane Gros (eVaderis).

This work has been published at the French Compas workshop [8] and the RSP Symposium [2].

7.6. Dataflow-explicit futures

Participant: Ludovic Henrio.

A future is a place-holder for a value being computed, and we generally say that a future is resolved when the associated value is computed. In existing languages futures are either implicit, if there is no syntactic or typing distinction between futures and non-future values, or explicit when futures are typed by a parametric type and dedicated functions exist for manipulating futures. We defined a new form of future, named data-flow explicit futures [38], with specific typing rules that do not use classical parametric types. The new futures allow at the same time code reuse and the possibility for recursive functions to return futures like with implicit futures, and let the programmer declare which values are futures and where synchronisation occurs, like with explicit futures. We prove that the obtained programming model is as expressive as implicit futures but exhibits a different behaviour compared to explicit futures. The current status of this work is the following:

- A paper showing formally the difference between implicit and explicit futures is under submission
- We are working with collaborators from University of Uppsala and University of Oslo on the design of programming constructs mixing implicit and dataflow-explicit futures
- Amaury Maillé will do his internship in the Cash team (advised by Matthieu Moy and Ludovic Henrio), working on an implementation of dataflow-explicit futures and further experiments with the model.

7.7. Locally abstract globally concrete semantics

Participant: Ludovic Henrio.

This research direction aims at designing a new way to write semantics for concurrent languages. The objective is to design semantics in a compositional way, where each primitive has a local behavior, and to adopt a style much closer to verification frameworks so that the design of an automatic verifier for the language is easier. The local semantics is expressed in a symbolic and abstract way, a global semantics gathers the abstract local traces and concretizes them. We have a reliable basis for the semantics of a simple language (a concurrent while language) and for a complex one (ABS), but the exact semantics and the methodology for writing it is still under development, we expect to submit a journal article during 2019 on the subject.

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

7.8. Memory consistency for heterogeneous systems

Participant: Ludovic Henrio.

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

- provided a formalization showing the correctness of VectorPU approach (published in 4PAD 2018, a symposium affiliated to HPCS).
- extended the work to support the manipulation of overlapping array (submitted as an extended version of the 4PAD paper)

We now plan to extend the work with support for concurrency.

7.9. PNets: Parametrized networks of automata

Participant: Ludovic Henrio.

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

- A weak bisimulation theory for open pNets is under development (a strong isimulation had already been defined in the past) and its properties are being proven, especially in terms of compositionality. This work is realized with Eric Madelaine (Inria Sophia-Antipolis) and Rabéa Ameur Boulifa (Telecom ParisTech).
- A translation from BIP model to open pNets is being formalized and encoded, this work is done in collaboration with Simon Bliudze (Inria Lille).

These works are under progress and should be continued in 2019.

7.10. Decidability results on the verification of phaser programs

Participant: Ludovic Henrio.

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

7.11. Practicing Domain-Specific Languages: From Code to Models

Participant: Laure Gonnord.

Together with Sebastien Mosser, we proposed a new Domain- Specific Language course at the graduate level whose objectives is to reconciliate concepts coming from Language Design as well as Modeling domains. We illustrate the course using the reactive systems application domain, which prevents us to fall back in a toy example pitfall. This paper describes the nine stages used to guide students through a journey starting at low-level C code to end with the usage of a language design workbench. This course was given as a graduate course available at Université Côte d'Azur (8 weeks, engineering-oriented) and École Normale Supérieure de Lyon (13 weeks, research-oriented).

The results have been published in a national software engineering conference [4] and the Models Educator Symposium [5].

7.12. Polyhedral Dataflow Programming: a Case Study

Participant: Laure Gonnord.

With Lionel Morel and Romain Fontaine (Insa Lyon), we have studied the benefits of jointly using polyhedral compilation with dataflow languages. We have proposed to expend the parallelization of dataflow programs by taking into account the parallelism exposed by loop nests describing the internal behavior of the program's agents. This approach is validated through the development of a prototype toolchain based on an extended version of the SigmaC language. We demonstrated the benefit of this approach and the potentiality of further improvements on several case studies.

The results have been published in the Sbac-PAD conference on High Performance computing [3].

7.13. Semantic Array Dataflow Analysis

Participants: Laure Gonnord, Paul Iannetta.

Together with Lionel Morel (Insa/CEA) and Tomofumi Yuki (Inria, Rennes), we revisited the polyhedral model's key analysis, dependency analysis. The semantic formulation we propose allows a new definition of the notion of dependency and the computation of the dependency set. As a side effect, we propose a general algorithm to compute *an over-approximation of* the dependency set of general imperative programs.

We argue that this new formalization will later allow for a new vision of the polyhedral model in terms of semantics, which will help us fully characterize its expressivity and applicability. We also believe that abstract semantics will be the key for designing an approximate abstract model in order to enhance the applicability of the polyhedral model.

The results is published in a research report [11].

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

Participant: Laure Gonnord.

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

The results have been accepted to VMCAI'19 on Model Checking and Abstract Interpretation [7].

7.15. Polyhedral Value Analysis as Fast Abstract Interpretation

Participant: Laure Gonnord.

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

We formulate the polyhedral value analysis (a classical algorithm in production compilers like LLVM, scalar evolution based on Presburger set as abstract interpretation), present a set of fast join operators, and show that aggressively falling back to top (rather than continuing with approximations) results in a scalable analysis. By formally describing the required analysis, we provide the necessary theoretical foundations for analysing large program systems with hundred thousands of loops and complex control flow structures at a precision high enough to cater for high-precision users such as polyhedral optimization frameworks, at a compile-time cost comparable with just compiling the application.

The paper is under redaction process.

8. Partnerships and Cooperations

8.1. National Initiatives

8.1.1. ANR

- Matthieu Moy submitted an ANR project as scientific coordinator entitled "Distributed Efficient Architecture for the Rapid (Co)simulation of Multiphysics Objects" (48 months, partners Verimag, TIMA and LIP6).
- Laure Gonnord's "Jeune Chercheur" ANR, CODAS, has started in January 2018 (42 months).

8.1.2. Scientific Advising

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

8.2. International Initiatives

8.2.1. Informal International Partners

- Christophe Alias has regular collaborations with Sanjay Rajopadhye from Colorado State University, USA (3 publications, one publication submitted 7.1).
- Ludovic Henrio has regular collaborations with university of Linköping (one publication last year, 2 submissions); University of Oslo, University of Uppsala, and TU Darmstadt on active objects (2 publications being written); Chalmers university and Univ of Twente (one publication submitted).
- Laure Gonnord has regular collaborations with Fernando Pereira from UFMG, Brasil (5 publications in total, last in 2017). In 2018 she has began a collaboration with Tobias Grösser, from ETH Zurich.

9. Dissemination

9.1. Promoting Scientific Activities

9.1.1. Member of the Organizing Committees

- Laure Gonnord animates the french compilation community since 2010 (http://compilfr.ens-lyon.fr
- Laure Gonnord has coorganised (with other members of the LIP lab) the doctoral school EJCP (École jeunes chercheur.se.s en programmation), in June 2018.

9.1.2. Chair of Conference Program Committees

- Ludovic Henrio was one of the chairs of the **ICE** 2018 workshop (Interaction and Concurrency Experiences), he will also be chairing **ICE** 2019.

9.1.3. Member of the Conference Program Committees

- Christophe Alias is a PC member of **HiP3ES** 2018 – 6th High Performance Energy Efficient Embedded Systems Workshop. He will be a PC member for **COMPAS** 2019 – Conférence d'informatique en Parallélisme, Architecture et Système.
- Matthieu Moy is a PC member of **DUHDe** 2018 — 5th Workshop on Design Automation for Understanding Hardware Designs.

- Laure Gonnord is a PC member of **TAPAS** 2018 - Ninth Workshop on Tools for Automatic Program Analysis, and **VMCAI** 2018 - 19th International Conference on Verification, Model Checking, and Abstract Interpretation. She will be a PC member of **CAV** 2019 Conference on Computer-Aided Verification.
- Ludovic Henrio is a PC member of **4PAD** 2018 (Formal Approaches to Parallel and Distributed Systems), **FASE** 2019 (Fundamental Approaches to Software Engineering). He will be a PC member of **Coordination** 2019 (Conference on Coordination Models and Languages), **ACSD** 2019 (Application of Concurrency to System Design).

9.1.4. Reviewer

- Christophe Alias was reviewer for **HCW** 2018 – 27th International Workshop on Heterogeneity in Computing.
- Matthieu Moy was reviewer for **DATE** 2018 - Design, Automation and Test in Europe, **MCSoC** 2019 - International Symposium on Embedded Multicore/Many-core Systems-on-Chip
- Laure Gonnord was reviewer for **STACS** 2018 - Symposium of Theorical Aspects of Computer Science.
- Ludovic Henrio was reviewer for **FM'2018**.

9.1.5. Journal

9.1.5.1. Reviewer - Reviewing Activities

- Christophe Alias was reviewer for ACM Transactions on Architectures and Code Optimization (**TACO**, ACM) and **Proceedings of the IEEE**.
- Matthieu Moy was reviewer for the Microprocessors and Microsystems Journal (**MICPRO**, Elsevier).
- Ludovic Henrio was reviewer for **JLAMP** (Journal of Logical and Algebraic Methods in Programming), Elsevier.

9.1.6. Invited Talks

- Matthieu Moy presented a talk "Response Time Analysis of Dataflow Applications on a Many-Core Processor with Shared-Memory and Network-on-Chip" for the 30 years of the LIP laboratory, November 2018.
- Laure Gonnord gave two invited talks at the workshop **FAC Days** (Toulouse) and at the International Conference **LOPSTR 2018** "Experiences in designing scalable static analyses".
- Ludovic Henrio gave an invited talk at **HLPP 2018** "An Overview of (some) Active-object Languages" and at **PASS** 2018 workshop "SafePlace: Trustable Virtual Machine Scheduling".

9.1.7. Research Administration

- Laure Gonnord is member of the doctoral commitee of the Inria Grenoble Rhône Alpes center.
- Laure Gonnord is elected member of the LIP council and the Fédération d'Informatique de Lyon council.

9.2. Teaching - Supervision - Juries

9.2.1. Teaching

Licence:

- Christophe Alias, Compilation, CM+TD, 27h, 3A, INSA Centre Val de Loire.
- Laure Gonnord, Algorithmic, C++ Programming, TP: L2, UCBL
- Laure Gonnord, Operating Systems, CM+TD+TP, 24h, L2, UCBL

- Laure Gonnord, Formal Languages, TD+TP, 15h, L3, UCBL
- Laure Gonnord, Logics, TD+TP, 18h, L3, UCBL.
- Matthieu Moy, Concurrent Programming, CM+TD+TP, 57h, L3, UCBL.
- Matthieu Moy, Recursive Programming, TD+TP, 48h, L1, UCBL.
- Matthieu Moy, Software Engineering, CM+TD+TP, 25h, M1, UCBL.
- Matthieu Moy, Git, CM+TP: 12h, L3, UCBL.
- Paul Iannetta, ACM, TD, 32h, L3, ENS de Lyon.
- Julien Braine, Théorie de la programmation, TD/TP, 32h, L3, ENS de Lyon.
- Julien Braine, Projet 1, TP, 32h, L3, ENS de Lyon.

Master:

- Christophe Alias, Compiler optimizations for embedded applications, CM+TD, 27h, 4A, INSA Centre Val de Loire.
- Christophe Alias and Matthieu Moy, Hardware Compilation and Simulation, CM+TD, 36h, M2 Informatique Fondamentale, ENS de Lyon.
- Laure Gonnord, Compilation and Program Analysis, CM, 28h, M1, ENS de Lyon.
- Laure Gonnord, Compilation and program transformations, CM+TD+TP, 35h, M1, UCBL.
- Laure Gonnord, Real Time Systems, CM+TD+TP, 30h, M1, UCBL.
- Laure Gonnord (25%), with Sebastien Mosser, Software Engineering and Compilation, CM+TP, 36h, M2 Informatique Fondamentale, ENS de Lyon.
- Laure Gonnord, Graphs, Complexity, Algorithmics, M1 MEEF (CAPES prepa), CM+TD+TP+oral training, 30h, UCBL.
- Matthieu Moy, Compilation and Program Analysis, TP, 16h, M1, ENS de Lyon.
- Matthieu Moy, Compilation and program transformations, TD+TP, 25h, M1, UCBL.
- Paul Iannetta, Projet intégré, 28h, M1, ENS de Lyon.
- Ludovic Henrio, Distributed Systems: an algorithmic approach, CM+TD, 7h, M2 Specialite IFI (Ingénierie et Fondements de l'Informatique), parcours CSSR, and UBINET, Université de Nice Sophia-Antipolis.

9.2.2. Supervision

- PhD: Amaury Graillat, "Parallel Code Generation of Synchronous Programs for a Many-core Architecture", Univ. Grenoble Alpes, defended on November 16th 2018, supervised by Matthieu Moy (LIP), Pascal Raymond (Verimag) and Benoît Dinechin (Kalray).
- PhD in progress: Gabriel Busnot, "Accélération SystemC pour la co-simulation multi-physique et la simulation de modèles hétérogènes en complexité", Univ. Lyon 1, started in october 2017, supervised by Matthieu Moy (LIP) and Tanguy Sassolas (CEA-LIST).
- PhD in progress: Tristan Delizy, "Dynamic Memory Management For Embedded Non-Volatile Memory", INSA Lyon, started in October 2016, supervised by Guillaume Salagnac (CITI), Tanguy Risset (CITI), Kevin Marquet (CITI) and Matthieu Moy (LIP).
- PhD in progress (from Sept. 2018): Paul Iannetta "Complex data structures scheduling for optimizing compilers", supervised by Lionel Morel (CITI/CEA) and Laure Gonnord (LIP).
- PhD in progress (from Sept. 2018): Julien Braine "Horn Clauses as an Efficient Intermediate Representation for Data Structure Verification", supervised by David Monniaux (CNRS/Verimag) and Laure Gonnord (LIP).
- PhD in progress: Pierre Leca, "Distributed BSP: Active Objects for BSPlib programs", CIFRE Huawei/UNS, started in August 2017, supervised by Gaëtan Hains (Huawei), Wijnand Suijlen (Huawei), Françoise Baude (UNS./I3S), Ludovic Henrio (LIP).

9.2.3. Juries

- Christophe Alias was an expert for the midterm PhD evaluation of Hang Yu from Université Grenoble-Alpes. Hang Yu is supervised by Michaël Perrin and David Monniaux.
- Laure Gonnord was an expert for the midterm PhD evaluation of Sébastien Bonnieux from University Nice Côte d'Azur. Sébastien Bonnieux was supervised by Sébastien Mosser and Mireille Blay-Fornarino.
- Laure Gonnord was reviewer for the PhD of Vincent Botbol from Sorbonne Université entitled "Analyse Statique de programmes concurrents avec variables numériques" and supervised by Emmanuel Chailloux and Tristan Le Gall.
- Laure Gonnord was external jury member for the PhD of Hoby Rakotoarivelo from Université Paris-Saclay entitled "Approche de co-design de noyaux irréguliers sur accélérateurs manycore. Application au cas du remaillage adaptatif pour le calcul intensif" and supervised by Franck Ledoux et Franck Pommereau.
- Laure Gonnord was local jury member for the PhD of Mohammed Amer from Université de Lyon entitled "Centralized Optimization of the Association in IEEE 802.11 Networks" and supervised by Anthony Busson and Isabelle Guérin-Lassous.
- Matthieu Moy was an expert for the midterm PhD evaluation of Joumana Lagha from Ecole centrale de Nantes. Joumana Lagha is supervised by Prof. Olivier H. Roux, Sebastien Faucou and Jean-luc Bechennec.
- Matthieu Moy was reviewer for the Ph.D of Benjamin Rouxel from Université de Rennes 1 entitled "Minimising communication costs impact when scheduling real-time applications on multi-core architectures" and supervised by Isabelle Puaut and Steven Derrien.

9.2.4. Internships

- Bilel Aouadhi, a last year engineer student from Faculté des sciences de Tunis, worked from April 2018 to July 2018 on the implementation of a visualization tool for Data-aware process networks. His was supervised by Christophe Alias and Matthieu Moy, his internship was funded by Université Lyon 1.
- Ligia Novacean, a L3 student from University of Cluj-Napoca (Romania), worked from July 2018 to September 2018 on a DPN-to-C translator for the HLS tool of Xilinx, VivadoHLS. She was supervised by Christophe Alias and Matthieu Moy, her internship was funded by Inria.
- Alexandra Dobre, a L3 student from University of Cluj-Napoca (Romania), worked from July 2018 to September 2018 on the generation of a SystemC simulator from a Data-aware process network. She was supervised by Matthieu Moy and Christophe Alias, her internship was funded by Inria.
- Arthur Gontier, a L3 student from University of Nantes, worked from April 2018 to July 2018 on the formal functional verification of Lustre code with Horn Clauses. He was supervised by Lionel Morel and Laure Gonnord. His internship was founded by the Codas ANR.
- Paul Iannetta, a M2 student from ENS de Lyon, worked from March 2018 to July 2018 on a semantic formalisation of the polyhedral model. He was supervised by Lionel Morel and Laure Gonnord. His internship was founded by the Codas ANR.

9.3. Popularization

9.3.1. Education

- Laure Gonnord is part of the local organisation of the Computer Science preparation for the Agregation examination for future maths teachers (MEEF).

9.3.2. Interventions

- Talk at “Campus du libre” by Matthieu Moy, Doua Lyon, “Pourquoi et comment se lancer dans le libre quand on est étudiant (ou pas) ?”, November 2018.

9.3.3. Internal action

- Café développeur by Matthieu Moy, “Utilisation avancée de Git” at LIRIS (2 sessions), October 2018.

9.3.4. Creation of media or tools for science outreach

- Video “Mon équipe en 180 secondes” by Matthieu Moy for the CASH team.

10. Bibliography

Publications of the year

International Conferences with Proceedings

[1] C. ALIAS. *Improving Communication Patterns in Polyhedral Process Networks*, in "HIP3ES 2018 - Sixth International Workshop on High Performance Energy Efficient Embedded Systems", Manchester, United Kingdom, January 2018, pp. 1-6, https://hal.inria.fr/hal-01725143

[2] T. DELIZY, S. GROS, K. MARQUET, M. MOY, T. RISSET, G. SALAGNAC. *Estimating the Impact of Architectural and Software Design Choices on Dynamic Allocation of Heterogeneous Memories*, in "RSP 2018 - 29th International Symposium on Rapid System Prototyping", Turin, Italy, October 2018, pp. 1-7, https://hal.archives-ouvertes.fr/hal-01891599

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

[4] L. GONNORD, S. MOSSER. *Du code aux modèles, des modèles au code: enseigner les langages dédiés (DSL)*, in "CIEL 2018 : 7ème Conférence en IngénieriE du Logiciel", Grenoble, France, June 2018, pp. 1-4, https://hal.archives-ouvertes.fr/hal-01816239

[5] L. GONNORD, S. MOSSER. *Practicing Domain-Specific Languages: From Code to Models*, in "14th Educators Symposium at MODELS 2018", Copenaghen, Denmark, October 2018, pp. 1-8 [*DOI :* 10.1145/3270112.3270116], https://hal.archives-ouvertes.fr/hal-01865448

[6] A. GRAILLAT, M. MOY, P. RAYMOND, B. DUPONT DE DINECHIN. *Parallel Code Generation of Synchronous Programs for a Many-core Architecture*, in "DATE 2018 - Design, Automation and Test in Europe", Dresden, Germany, IEEE, March 2018, pp. 1139-1142 [*DOI :* 10.23919/DATE.2018.8342182], https://hal.inria.fr/hal-01667594

Conferences without Proceedings

[7] C. BALLABRIGA, J. FORGET, L. GONNORD, G. LIPARI, J. RUIZ. *Static Analysis Of Binary Code With Memory Indirections Using Polyhedra*, in "International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI'19)", Lisbon, Portugal, January 2019, https://hal.archives-ouvertes.fr/hal-01939659

[8] T. DELIZY, S. GROS, K. MARQUET, M. MOY, T. RISSET, G. SALAGNAC. *Quels objets en NVRAM ? Placement en mémoires de travail hétérogènes*, in "Compas 2018 - Conférence d'informatique en Parallélisme, Architecture et Système", Toulouse, France, July 2018, pp. 1-8, https://hal.archives-ouvertes.fr/hal-01891398

Research Reports

[9] C. ALIAS. *FIFO Recovery by Depth-Partitioning is Complete on Data-aware Process Networks*, Inria Grenoble - Rhone-Alpes, June 2018, n° RR-9187, https://hal.inria.fr/hal-01818585

[10] L. GONNORD, P. IANNETTA, L. MOREL. *Semantic Polyhedral Model for Arrays and Lists*, Inria Grenoble Rhône-Alpes ; UCBL ; LIP - ENS Lyon ; CEA List, June 2018, n° RR-9183, pp. 1-28, https://hal.archives-ouvertes.fr/hal-01815759

[11] P. IANNETTA, L. GONNORD, L. MOREL, T. YUKI. *Semantic Array Dataflow Analysis*, Inria Grenoble Rhône-Alpes, December 2018, n° RR-9232, https://hal.archives-ouvertes.fr/hal-01954396

[12] G. IOOSS, C. ALIAS, S. RAJOPADHYE. *Monoparametric Tiling of Polyhedral Programs*, Inria Grenoble - Rhone-Alpes, December 2018, https://hal.inria.fr/hal-01952593

References in notes

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

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

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

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

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

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

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

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

[21] C. B. AOUN, L. ANDRADE, T. MAEHNE, F. PÊCHEUX, M.-M. LOUËRAT, A. VACHOUXY. *Pre-simulation elaboration of heterogeneous systems: The SystemC multi-disciplinary virtual prototyping approach*, in

"Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS), 2015 International Conference on", IEEE, 2015, pp. 278–285

[22] P. AUBRY, P.-E. BEAUCAMPS, F. BLANC, B. BODIN, S. CARPOV, L. CUDENNEC, V. DAVID, P. DORÉ, P. DUBRULLE, B. DUPONT DE DINECHIN, F. GALEA, T. GOUBIER, M. HARRAND, S. JONES, J.-D. LESAGE, S. LOUISE, N. MOREY CHAISEMARTIN, T. H. NGUYEN, X. RAYNAUD, R. SIRDEY. *Extended Cyclostatic Dataflow Program Compilation and Execution for an Integrated Manycore Processor*, in "Alchemy 2013 - Architecture, Languages, Compilation and Hardware support for Emerging ManYcore systems", Barcelona, Spain, Proceedings of the International Conference on Computational Science, ICCS 2013, June 2013, vol. 18, pp. 1624-1633 [*DOI :* 10.1016/J.PROCS.2013.05.330], https://hal.inria.fr/hal-00832504

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

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

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

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

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

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

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

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

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

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

[33] R. FONTAINE, L. GONNORD, L. MOREL. *Polyhedral Dataflow Programming: a Case Study*, in "International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)", IEEE, September 2018

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

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

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

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

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

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

[40] A. KRIZHEVSKY, I. SUTSKEVER, G. E. HINTON. *Imagenet classification with deep convolutional neural networks*, in "Advances in neural information processing systems", 2012, pp. 1097–1105

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

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

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

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

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

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

[47] W. H. PRESS, S. A. TEUKOLSKY, W. T. VETTERLING, B. P. FLANNERY. *Numerical recipes in C++*, in "The art of scientific computing", 2015

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

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

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

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

[52] J. TRAVIS, J. KRING. *LabVIEW for everyone: graphical programming made easy and fun*, Prentice-Hall, 2007

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

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

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

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

[57] P. WILMOTT. *Quantitative Finance*, Wiley, 2006

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