RESEARCH CENTRE

Paris

IN PARTNERSHIP WITH:

CNRS, Ecole normale supérieure de Paris

2020

ACTIVITY REPORT

Project-Team PARKAS

Parallélisme de Kahn Synchrone

IN COLLABORATION WITH: Département d'Informatique de l'Ecole Normale Supérieure

DOMAIN

Algorithmics, Programming, Software and Architecture

THEME

Embedded and Real-time Systems

Contents

Project-Team PARKAS

Creation of the Team: 2011 April 01, updated into Project-Team: 2012 January 01

Keywords

Computer sciences and digital sciences

A1.1.1. – Multicore, Manycore

A1.2.7. – Cyber-physical systems

A2.1.1. – Semantics of programming languages

A2.1.4. – Functional programming

A2.1.6. – Concurrent programming

A2.1.9. – Synchronous languages

A2.1.10. – Domain-specific languages

A2.2.4. – Parallel architectures

A2.2.8. – Code generation

A2.3. – Embedded and cyber-physical systems

A2.3.1. – Embedded systems

A2.3.2. – Cyber-physical systems

A2.3.3. – Real-time systems

A2.4.3. – Proofs

A3.4.5. – Bayesian methods

A6.2.1. – Numerical analysis of PDE and ODE

A6.2.2. – Numerical probability

A6.2.3. – Probabilistic methods

A6.4.1. – Deterministic control

A6.4.2. – Stochastic control

Other research topics and application domains

B5.2.1. – Road vehicles

B5.2.2. – Railway

B5.2.3. – Aviation

B6.4. – Internet of things

B6.6. – Embedded systems

B7.2.1. – Smart vehicles

B9.5.1. – Computer science

B9.5.2. – Mathematics

1 Team members, visitors, external collaborators

Research Scientists

- Guillaume Baudart [Inria, from Oct 2020, Starting Faculty Position]
- Timothy Bourke [Inria, Researcher]

Faculty Members

- Marc Pouzet [Team leader, Sorbonne Université, Professor, HDR]
- Paul Feautrier [Université de Lyon, Emeritus, from May 2020]

Post-Doctoral Fellow

- Guillaume Iooss [Inria, until Jan 2020]

PhD Students

- Paul Jeanmaire [Inria, from Oct 2020]
- Ismail Lahkim Bennani [Inria]
- Baptiste Pauget [Inria, CIFRE, from Nov 2020]
- Basile Pesin [Inria, from Sep 2020]

Technical Staff

- Lelio Brun [Inria, Engineer]
- Andi Drebes [Inria, Engineer, until Aug 2020]

Interns and Apprentices

- Marc Coudriau [École Normale Supérieure de Paris, from Feb 2020 until Jul 2020]
- Baptiste Pauget [Inria, from Feb 2020 until Jul 2020]
- Basile Pesin [Inria, from Mar 2020 until Aug 2020]

Administrative Assistants

- Christine Anocq [Inria]
- Nelly Maloisel [Inria]

2 Overall objectives

Research in PARKAS focuses on the design, semantics, and compilation of programming languages which allow going from parallel deterministic specifications to target embedded code executing on sequential or multi-core architectures. We are driven by the ideal of a mathematical and executable language used both to program and simulate a wide variety of systems, including real-time embedded controllers in interaction with a physical environment (e.g., fly-by-wire, engine control), computationally intensive applications (e.g., video), and compilers that produce provably correct and efficient code.

The team bases its research on the foundational work of Gilles Kahn on the semantics of deterministic parallelism, the theory and practice of synchronous languages and typed functional languages, synchronous circuits, modern (polyhedral) compilation, and formal models to prove the correctness of low-level code.

To realize our research program, we develop languages (LUCID SYNCHRONE, REACTIVEML, LUCY-N, ZELUS), compilers, contributions to open-source projects (Sundials/ML), and formalizations in Interactive Theorem Provers of language semantics (Vélus and *n*-synchrony). These software projects constitute essential “laboratories”: they ground our scientific contributions, guide and validate our research through experimentation, and are an important vehicle for long-standing collaborations with industry.

3 Research program

3.1 Programming Languages for Cyber-Physical Systems

We study the definition of languages for reactive and Cyber-Physical Systems in which distributed control software interacts closely with physical devices. We focus on languages that mix discrete-time and continuous-time; in particular, the combination of synchronous programming constructs with differential equations, relaxed models of synchrony for distributed systems communicating via periodic sampling or through buffers, and the embedding of synchronous features in a general purpose ML language.

The synchronous language SCADE,[1] based on synchronous languages principles, is ideal for programming embedded software and is used routinely in the most critical applications. But embedded design also involves modeling the control software together with its environment made of physical devices that are traditionally defined by differential equations that evolve on a continuous-time basis and approximated with a numerical solver. Furthermore, compilation usually produces single-loop code, but implementations increasingly involve multiple and multi-core processors communicating via buffers and shared-memory.

The major player in embedded design for cyber-physical systems is undoubtedly SIMULINK,[2] with MODELICA[3] a new player. Models created in these tools are used not only for simulation, but also for test-case generation, formal verification, and translation to embedded code. That said, many foundational and practical aspects are not well-treated by existing theory (for instance, hybrid automata), and current tools. In particular, features that mix discrete and continuous time often suffer from inadequacies and bugs. This results in a broken development chain: for the most critical applications, the model of the controller must be reprogrammed into either sequential or synchronous code, and properties verified on the source model have to be reverified on the target code. There is also the question of how much confidence can be placed in the code used for simulation.

We attack these issues through the development of the ZELUS[4] research prototype, industrial collaborations with the SCADE team at ANSYS/Esterel-Technologies, and collaboration with Modelica developers at Dassault-Systèmes and the Modelica association. Our approach is to develop a *conservative extension* of a synchronous language capable of expressing in a single source text a model of the control software and its physical environment, to simulate the whole using off-the-shelf numerical solvers, and to generate target embedded code. Our goal is to increase faithfulness and confidence in both what is actually executed on platforms and what is simulated. The goal of building a language on a strong mathematical basis for hybrid systems is shared with the Ptolemy project at UC Berkeley; our approach is distinguished by building our language on a synchronous semantics, reusing and extending classical synchronous compilation techniques.

Adding continuous time to a synchronous language gives a richer programming model where reactive controllers can be specified in idealized physical time. An example is the so called quasi-periodic architecture studied by Caspi, where independent processors execute periodically and communicate by sampling. We have applied ZELUS to model a class of quasi-periodic protocols and to analyze an abstraction proposed for model-checking such systems.

[1] http://www.esterel-technologies.com/products/scade-suite
[2] http://www.mathworks.com/products/simulink
[3] https://www.modelica.org
[4] https://zelus.di.ens.fr

Communication-by-sampling is suitable for control applications where value timeliness is paramount and lost or duplicate values tolerable, but other applications—for instance, those involving video streams—seek a different trade-off through the use of bounded buffers between processes. We developed the n-synchronous model and the programming language LUCY-N to treat this issue.

3.2 Compiling for Sequential and Multi-Core Processors

We develop compilation techniques for sequential and multi-core processors, and efficient parallel run-time systems for computationally intensive real-time applications (e.g., video and streaming). We study the generation of parallel code from synchronous programs, compilation techniques based on the polyhedral model, and the exploitation of synchronous Single Static Assignment (SSA) representations in general purpose compilers.

We consider distribution and parallelism as two distinct concepts.

- Distribution refers to the construction of multiple programs which are dedicated to run on specific computing devices. When an application is designed for, or adapted to, an embedded multiprocessor, the distribution task grants fine grained—design- or compilation-time—control over the mapping and interaction between the multiple programs.
- Parallelism is about generating code capable of efficiently exploiting multiprocessors. Typically this amounts to making (in)dependence properties, data transfers, atomicity and isolation explicit. Compiling parallelism translates these properties into low-level synchronization and communication primitives and/or onto a runtime system.

We also see a strong relation between the foundations of synchronous languages and the design of compiler intermediate representations for concurrent programs. These representations are essential to the construction of compilers enabling the optimization of parallel programs and the management of massively parallel resources. Polyhedral compilation is one of the most popular research avenues in this area. Indirectly, the design of intermediate representations also triggers exciting research on dedicated runtime systems supporting parallel constructs. We are particularly interested in the implementation of non-blocking dynamic schedulers interacting with decoupled, deterministic communication channels to hide communication latency and optimize local memory usage.

While distribution and parallelism issues arise in all areas of computing, our programming language perspective pushes us to consider four scenarios:

1. designing an embedded system, both hardware and software, and codesign;
2. programming existing embedded hardware with functional and behavioral constraints;
3. programming and compiling for a general-purpose or high-performance, best-effort system;
4. programming large scale distributed, I/O-dominated and data-centric systems.

We work on a multitude of research experiments, algorithms and prototypes related to one or more of these scenarios. Our main efforts focused on extending the code generation algorithms for synchronous languages and on the development of more scalable and widely applicable polyhedral compilation methods.

3.3 Validation and Proof of Compilers

Compilers are complex software and not immune from bugs. We work on validation and proof tools for compilers to relate the semantics of source programs with the corresponding executable code.

The formal validation of a compiler for a synchronous language, or more generally for a language based on synchronous block diagrams, promises to reduce the likelihood of compiler-introduced bugs, the cost of testing, and also to ensure that properties verified on the source model hold of the target code. Such a validation would be complementary to existing industrial qualifications which certify the development process and not the functional correctness of a compiler. The scientific interest is in developing models and techniques that both facilitate the verification and allow for convenient reasoning over the semantics of a language and the behavior of programs written in it.

3.4 Probabilistic Reactive Programming

Most embedded systems evolve in an open, noisy environment that they only perceive through noisy sensors (e.g., accelerometers, cameras, or GPS). Another level of uncertainty comes from interactions with other autonomous entities (e.g., surrounding cars, or pedestrians crossing the street). Yet, to date, existing tools for cyber-physical system have had limited support for modeling uncertainty, to simulate the behavior of the systems, or to infer parameters from noisy observations. The classic approach consists in hand-coding robust stochastic controllers. But this solution is limited to well-understood and relatively simple tasks like the *lane following assist* system. However, no such controller can handle, for example, the difficult to anticipate behavior of a pedestrian crossing the street. A modern alternative is to rely on deep-learning techniques. But neural networks are black-box models that are notoriously difficult to understand and verify. Training them requires huge amounts of curated data and computing resources which can be problematic for corner-case scenarios in embedded control systems.

Over the last few years, Probabilistic Programming Languages (PPL) have been introduced to describe probabilistic models and automatically infer distributions of parameters from observed data. Compared to deep-learning approaches, probabilistic models show great promise: they overtly represent uncertainty, and they enable explainable models that can capture both expert knowledge and observed data.

A probabilistic reactive language provides the facilities of a synchronous language to write control software, with probabilistic constructs to model uncertainties and perform inference-in-the-loop. This approach offers two key advantages for the design of embedded systems with uncertainty: 1) Probabilistic models can be used to simulate an uncertain environment for early stage design and incremental development. 2) The embedded controller itself can rely on probabilistic components which implement skills that are out of reach for classic automatic controllers.

4 Application domains

4.1 Embedded Control Software

Embedded control software defines the interactions of specialized hardware with the physical world. It normally ticks away unnoticed inside systems like medical devices, trains, aircraft, satellites, and factories. This software is complex and great effort is required to avoid potentially serious errors, especially over many years of maintenance and reuse.

Engineers have long designed such systems using block diagrams and state machines to represent the underlying mathematical models. One of the key insights behind synchronous programming languages is that these models can be executable and serve as the base for simulation, validation, and automatic code generation. This approach is sometimes termed Model-Based Development (MBD). The SCADE language and associated code generator allow the application of MBD in safety-critical applications. They incorporate ideas from LUSTRE, LUCID SYNCHRONE, and other programming languages.

4.2 Hybrid Systems Design and Simulation

Modern embedded systems are increasingly conceived as rich amalgams of software, hardware, networking, and physical processes. The terms Cyberphysical System (CPS) or Internet-of-Things (IoT) are sometimes used as labels for this point of view.

In terms of modeling languages, the main challenges are to specify both discrete and continuous processes in a single *hybrid* language, give meaning to their compositions, simulate their interactions, analyze the behavior of the overall system, and extract code either for target control software or more efficient, possibly online, simulation. Languages like Simulink and Modelica are already used in the design and analysis of embedded systems; it is more important than ever to understand their underlying principles and to propose new constructs and analyses.

5 Highlights of the year

The PARKAS team organized the 27th International Open Workshop on Synchronous Programming (SYNCHRON 2020).[5]

5.1 Awards

Guillaume Baudart and his co-authors received an *ACM SIGSOFT Distinguished Paper Award* for "A Principled Approach to GraphQL Query Cost Analysis" [15] at the ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE) in November 2020.

6 New software and platforms

6.1 New software

6.1.1 Heptagon

Keywords: Compilers, Synchronous Language, Controller synthesis

Functional Description: Heptagon is an experimental language for the implementation of embedded real-time reactive systems. It is developed inside the Synchronics large-scale initiative, in collaboration with Inria Rhones-Alpes. It is essentially a subset of Lucid Synchrone, without type inference, type polymorphism and higher-order. It is thus a Lustre-like language extended with hierchical automata in a form very close to SCADE 6. The intention for making this new language and compiler is to develop new aggressive optimization techniques for sequential C code and compilation methods for generating parallel code for different platforms. This explains much of the simplifications we have made in order to ease the development of compilation techniques.

The current version of the compiler includes the following features: - Inclusion of discrete controller synthesis within the compilation: the language is equipped with a behavioral contract mechanisms, where assumptions can be described, as well as an "enforce" property part. The semantics of this latter is that the property should be enforced by controlling the behaviour of the node equipped with the contract. This property will be enforced by an automatically built controller, which will act on free controllable variables given by the programmer. This extension has been named BZR in previous works. - Expression and compilation of array values with modular memory optimization. The language allows the expression and operations on arrays (access, modification, iterators). With the use of location annotations, the programmer can avoid unnecessary array copies.

URL: `http://heptagon.gforge.inria.fr`

Contacts: Marc Pouzet, Adrien Guatto, Gwenaël Delaval

Participants: Adrien Guatto, Brice Gelineau, Cédric Pasteur, Eric Rutten, Gwenaël Delaval, Léonard Gérard, Marc Pouzet

Partners: UGA, ENS Paris, Inria, LIG

6.1.2 SundialsML

Name: Sundials/ML

Keywords: Simulation, Mathematics, Numerical simulations

Scientific Description: Sundials/ML is a comprehensive OCaml interface to the Sundials suite of numerical solvers (CVODE, CVODES, IDA, IDAS, KINSOL). Its structure mostly follows that of the Sundials

[5] `https://synchron2020.inria.fr`

library, both for ease of reading the existing documentation and for adapting existing source code, but several changes have been made for programming convenience and to increase safety, namely:

solver sessions are mostly configured via algebraic data types rather than multiple function calls,

errors are signalled by exceptions not return codes (also from user-supplied callback routines),

user data is shared between callback routines via closures (partial applications of functions),

vectors are checked for compatibility (using a combination of static and dynamic checks), and

explicit free commands are not necessary since OCaml is a garbage-collected language.

Functional Description: Sundials/ML is a comprehensive OCaml interface to the Sundials suite of numerical solvers (CVODE, CVODES, IDA, IDAS, KINSOL, ARKODE).

Release Contributions: Adds support for v4.x of the Sundials Suite of numerical solvers.

Notably this release adds support for nonlinear solvers, improves the interface to linear solvers, adds support for new vector array operations, and subdivides the ARKode interface into three submodules (ARKStep, ERKStep, and MRIStep).

URL: `http://inria-parkas.github.io/sundialsml/`

Publications: hal-01408230v1, hal-01967659v1

Authors: Jun Inoue, Timothy Bourke, Marc Pouzet

Contacts: Marc Pouzet, Timothy Bourke

Participants: Jun Inoue, Marc Pouzet, Timothy Bourke

6.1.3 Zelus

Keywords: Numerical simulations, Compilers, Embedded systems, Hybrid systems

Scientific Description: The Zélus implementation has two main parts: a compiler that transforms Zélus programs into OCaml programs and a runtime library that orchestrates compiled programs and numeric solvers. The runtime can use the Sundials numeric solver, or custom implementations of well-known algorithms for numerically approximating continuous dynamics.

Functional Description: Zélus is a new programming language for hybrid system modeling. It is based on a synchronous language but extends it with Ordinary Differential Equations (ODEs) to model continuous-time behaviors. It allows for combining arbitrarily data-flow equations, hierarchical automata and ODEs. The language keeps all the fundamental features of synchronous languages: the compiler statically ensure the absence of deadlocks and critical races, it is able to generate statically scheduled code running in bounded time and space and a type-system is used to distinguish discrete and logical-time signals from continuous-time ones. The ability to combines those features with ODEs made the language usable both for programming discrete controllers and their physical environment.

URL: `https://zelus.di.ens.fr`

Publications: hal-03051954v1, hal-02333603v1, hal-02426533v1, inria-00554271v1, hal-01242732v1, hal-00654113v1, hal-00909029v1, hal-01575621v4, hal-01575631v1, hal-00766726v1, hal-00938891v1, hal-00654112v1, hal-01879026v1, hal-01549183v2, hal-00938866v1

Authors: Marc Pouzet, Timothy Bourke

Contacts: Marc Pouzet, Timothy Bourke, Guillaume Baudart

Participants: Marc Pouzet, Timothy Bourke

Partner: ENS Paris

6.1.4 Vélus

Name: Verified Lustre Compiler

Keywords: Synchronous Language, Compilation, Software Verification, Coq, Ocaml

Functional Description: Vélus is a prototype compiler from a subset of Lustre to assembly code. It is written in a mix of Coq and OCaml and incorporates the CompCert verified C compiler. The compiler includes formal specifications of the semantics and type systems of Lustre, as well as the semantics of intermediate languages, and a proof of correctness that relates the high-level dataflow model to the values produced by iterating the generated assembly code.

Release Contributions: First source-code release. Treatment of primitive reset construct. Clocks allowed for node arguments.

URL: `https://velus.inria.fr`

Contacts: Timothy Bourke, Lelio Brun, Marc Pouzet

6.1.5 MPPcodegen

Name: Source-to-source loop tiling based on MPP

Keywords: Source-to-source compiler, Polyhedral compilation

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

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

Publication: hal-02493164

Authors: Christophe Alias, Guillaume Iooss, Sanjay Rajopadhye

Contacts: Christophe Alias, Guillaume Iooss, Sanjay Rajopadhye

Partner: Colorado State University

6.1.6 MPP

Name: MonoParametric Partitionning transformation

Keywords: Compilation, Polyhedral compilation

Functional Description: This library applies a monoparametric partitioning transformation to polyhedra and affine functions. This transformation is a subset of the parametric sized tiling transformation, specialized for the case where shapes depend only on a single parameter. Unlike in the general case, the resulting sets and functions remain in the polyhedral model.

URL: `https://github.com/guillaumeiooss/MPP`

Contacts: Guillaume Iooss, Christophe Alias, Sanjay Rajopadhye

6.1.7 ProbZelus

Keywords: Synchorous language, Probability

Functional Description: ProbZelus is a synchronous probabilistic programming language built on top of Zelus a dataflow language à la Scade/Lustre. ProZelus offers several streaming inference techniques including a semi-symbolic inference algorithm based on delayed sampling.

URL: `https://github.com/IBM/probzelus`

Authors: Guillaume Baudart, Louis Mandel, Eric Atkinson, Benjamin Sherman, Marc Pouzet, Michael Carbin

Contact: Guillaume Baudart

Partners: CSAIL, MIT, IBM

7 New results

7.1 Verified compilation of Lustre

Participants Timothy Bourke, Lélio Brun, Paul Jeanmaire, Basile Pesin, Marc Pouzet.

Vélus[6] is a compiler for a subset of LUSTRE and SCADE that is specified in the Coq [29] Interactive Theorem Prover (ITP). It integrates the CompCert C compiler [33, 24] to define the semantics of machine operations (integer addition, floating-point multiplication, etcetera) and to generate assembly code for different architectures. The research challenges are to

- to mechanize, i.e., put into Coq, the semantics of the programming constructs used in modern languages for Model-Based Development;
- to implement compilation passes and prove them correct;
- to interactively verify source programs and guarantee that the obtained invariants also hold of the generated code.

Work continued this year on this long-running project in three main directions: the wrapping up of L. Brun's thesis work, the addition of enumerated types to the compiler, and B. Pesin's M2 internship on normalizing Lustre.

Specifying and compiling the modular reset construct in Coq. In the original LUSTRE language, the only way to reset the internal state of an instantiated function is to propagate and test explicit reset signals. Later languages, like LUCID SYNCHRONE and SCADE, provide a construct for resetting an instance modularly (it works for any function) and efficiently (testing occurs only at the point of instantiation). L. Brun's thesis work focused on formalizing and compiling this construct in Coq with end-to-end correctness proofs. He introduced a novel semantic rule for adding the reset construct to an existing language, a new intermediate language that allows sequenced manipulations of shared state, and a proof based on making explicit intermediate memory manipulations. This work was presented in January at the 47th ACM SIGPLAN Symposium on Principles of Programming Languages (POPL 2020) in New Orleans [12]. After an initial cancellation due to the first lockdown, L. Brun defined his thesis in July [20].

[6] `https://velus.inria.fr`

Adding enumerated types Modern dataflow languages, like LUCID SYNCHRONE and SCADE, allow users to declare enumerated types and use them in constructions for sampling and conditional activation. This feature is not only useful in itself, but also provides a target for the compilation of advanced features like state machines [28]. This year, T. Bourke and L. Brun added enumerated types to the Vélus compiler. This involved generalizing the syntax and semantics of the `if`, `merge`, and `when` operators; adapting passes for optimizing and initializing sequential code; and adapting the interface with CompCert. We replaced the hard-coded `bool` type with a pre-declared enumerated type of two elements. This simplifies the semantics and compilation passes, but requires extra work to treat standard operators like `not`, `and`, and `or`, which are inefficient if implemented by branching statements rather than machine operations. This work provides a solid basis for the thesis work of B. Pesin and will eventually appear in the publicly available compiler.

Non-normalized Lustre. Much of our previous work has focused on a subset of "normalized" programs where the form of expressions and equations is constrained to facilitate compilation. Last year, during P. Jeanmaire's M2 internship, we generalized the definitions of syntax and semantics in our prototype compiler to accept non-normalized programs, and added a compilation pass called *transcription* to convert a normalized program in this richer syntax into the syntax used by downstream passes. This year, during B. Pesin's M2 internship, we implemented the passes to normalize programs in the richer syntax. Our implementation operates in two phases. The first simplifies the abstract syntax tree by unnesting delay operators and function instances, and distributing other operators over lists. The second replaces initialization expressions in delay operators by constant terms, introducing extra registers per clock rate to determine when initialization should occur. The correctness proofs involved showing that static clock annotations corresponded with the semantics of the program, and also that a certain notion of variable dependency is preserved by compilation passes. A report on this work was accepted to appear in the 32nd edition of the *Journées Francophones des Langages Applicatifs* (JFLA 2021).

Glossary

Interactive Theorem Prover (ITP, also known as a *proof assistant*) Software for formal specification and proof, with features for generating and checking proofs, and extracting programs for later compilation

Model-Based Development (MBD) The specification of control software using block-diagrams, state machines, and other high-level constructions allowing programmers to focus on describing desired behaviour and to rely on automatic code generation to produce low-level executables.

7.2 Latency-based scheduling of synchronous programs

Participants Timothy Bourke, Guillaume Iooss, Baptiste Pauget, Marc Pouzet.

External collaborators: Michel Angot, Vincent Bregeon, Jean Souyris, and Matthieu Boitrel, (Airbus).

It is sometimes desirable to compile a single synchronous language program into multiple tasks for execution by a real-time operating system. We have been investigating this question from three different perspectives.

Harmonic clocks We studied the extension of a synchronous language with periodic harmonic clocks based on the work of Mandel et al. [26, 36, 27, 34, 35] on n-synchrony and the extension proposed by Forget et al. [30]

Mandel et al. considered a language with periodic clocks expressed as ultimately periodic binary sequences. The decision procedures (equality, inclusion, precedence) for such an expressive language can be very costly. It is thus sometimes useful to apply an envelope-based abstraction, that is, one where sets of clocks are represented by a rational slope and an interval. Forget considered simpler "harmonic" clocks. His decision procedures conincide with those for the envelope-based abstraction but without any loss of information. During his M2 internship, B. Pauget continued this line of work by extending the input language of the Vélus Lustre compiler with harmonic clocks. This work was the starting point for the proposal of a new intermediate language for a synchronous compiler that is capable of exploiting clock information to apply agressive optimizations and generate parallel code.

New Intermediate Language MObc (Multi Object Code) This intermediate language is reminiscent of the intermediate Obc language used in the Vélus and Heptagon compiler, but with some important differences and new features. MObc permits a synchronous function to be represented as a set of named state variables and possibly nested blocks with a partial ordering which express the way blocks can and must be called. In comparison, Obc represents a synchronous function as a set of state variables and a transition function that is itself written in a sequential language. Each block comprises a set of equations in Single Static Assignment (SSA) form, that is, exactly one equation per variable, so as to simplify the implementation of a number of classic optimizations (for example, constant propagation, inlining, common sub-expression elimination, code specialisation). Then, every block is translated into a step function (e.g., a C function). This intermediate language has been designed to facilitate the generation of code for a real-time OS and a multi-core target. This work exploits two older results: the article of Caspi et al. [25] that introduces an object representation for synchronous nodes and a "scheduling policy" that specifies how their methods may be called, and; the work of Pouzet et al. [37] on the calculation of input/output relations to merge calculations. We are preparing and article on this subject.

Scheduling and code generation for periodic streams In this approach, the top-level node of a Lustre program is distinguished from inner nodes. It may contain special annotations to specify the triggering and other details of node instances from which separate "tasks" are to be generated. Special operators are introduced to describe the buffering between top-level instances. Notably, different forms of the `when` and `current` operators are provided. Some of the operators are under-specified and a constraint solver is used to determine their exact meaning, that is, whether the signal is delayed by zero, one, or more cycles of the receiving clock, which depends on the scheduling of the source and destination nodes. Scheduling is formalized as a constraint solving problem based on latency constraints between some pairs of input/outputs that are specified by the designer.

G. Iooss prototyped these ideas in the Heptagon compiler. He implemented a code generation scheme based on translating the extended operators into combinations of standard LUSTRE operators so as to reuse the existing compiler backend. While we learned much from these experiments, they also revealed that the code generated in this way tended to contain very many nested branching structures. This work was described in [22].

Continuing this work, and building on ideas from Prelude [31] and the n-synchronous model [34], T. Bourke began work on a new prototype compiler for a simplified version of the source language. The new source language is based on two main design designs. First, clocks no longer contain a phase component, they simply specify the rate (the inverse of the period). This simplifies the definition of clock equality and the types of sampling operators. Second, explicit buffer operators are no longer required. Rather, the notion "synchronous" flows is relaxed and more emphasis is placed on causality. These two changes lead to a new compilation scheme based on periodically writing and reading shared variables.

This work is funded by a direct industrial contract with Airbus.

7.3 Sundials/ML: OCaml interface to Sundials Numeric Solvers

Participants Timothy Bourke.

This year we made major updates to the Sundials/ML OCaml interface[7] to support v4.x of the Sundials Suite of numerical solvers.

[7] https://inria-parkas.github.io/sundialsml/

Notably this release adds support for nonlinear solvers, improves the interface to linear solvers, adds support for new vector array operations, and subdivides the ARKode interface into three submodules (ARKStep, ERKStep, and MRIStep). This work required the addition of support for bigarrays to X. Leroy's ocamlmpi library.[8]

7.4 The Zelus Language

Participants Guillaume Baudart, Ismail Lakhim-Bennani, Marc Pouzet.

Zelus is our laboratory to experiment our research on programming languages for hybrid systems. It is devoted to the design and implementation of systems that may mix discrete-time/continuous-time signals and systems between those signals. It is essentially a synchronous language reminiscent of Lustre and Lucid Synchrone but with the ability to define functions that manipulate continuous-time signals defined by Ordinary Differential Equations (ODEs). The language is functional in the sense that a system is a function from signals to signals (not a relation). It provides some features from ML languages like higher-order and parametric polymorphism as well as dedicated static analyses.

Distribution of the language The language, its compiler and examples (release 2.1) are now on GitHub: `https://github.com/INRIA/zelus`. It is also available as an OPAM package. All the installation machinery has been greatly simplified.

Set-based simulation of Zelus programs In collaboration with Francois Bidet (PhD. student under the supervision of Sylvie Putot and Eric Goubault from Ecole polytechnique), we are developing a method to perform set-based simulation of Zelus program. Set-based simulation goes beyond concrete simulation (the default simulation mode of all existing hybrid system modeling languages). Instead of computing one trajectory, it computes a set of trajectories or *flowpipes* at once, replacing a possibly unbounded number of concrete simulations. It is also able to deal with models with partially known parameters and inputs.

Very little tools currently deal with models expressed modularly (as the parallel and hierarchical composition of subsystems, with function application and the mix between a software model and ODEs, for example). A prototype is under way. Set based simulation is done on the intermediate language generated by Zelus, that is a collection of tarnsition functions acting on a state.

Property Based Testing of Hybrid Programs Property-based program testing involves checking an executable specification by running many tests. We build on the work of Georgios Fainekos and Alexandre Donzé, and take inspiration from earlier work by Nicolas Halbwachs, to write a Zélus library of synchronous observers with a quantitative semantics that can be used to specify properties of a system under test. We implemented several optimization algorithms for producing test cases, some of which are gradient-based. This year, we have studied the use SUNDIALS CVODEs (sensitivity analysis) to find more falsification examples and faster.

7.5 An executable reference semantics for Zelus

During year 2020, we have worked on the definition of a comprehensive semantics for Zelus language, including all language constructs, that is executable and can lead to a reference interpreter.

The scientific objective is to use it to test an existing compiler, to prove the correctness of compile-time checks (e.g., that a well typed/causal/initialized program does not lead to an error); to prove the semantics preservation of compiler transformations (e.g., static scheduling, compilation of automata); to execute unfinished programs or programs that are semantically correct but are statically rejected by the compiler. Examples are cyclic circuits accepted by an Esterel compiler (the so-called "constructively

[8]`https://github.com/xavierleroy/ocamlmpi`

causal" programs) but are rejected by Lustre, Lucid Synchrone, Scade, Zelus compilers that impose stronger causality constraints; finally to prototype new language constructs.

The existing semantics for rich languages like Scade is defined by its translation into a small data-flow language; we expect instead to have a semantics that apply directly to the source, before any rewritting or check is made.

The current prototype we have developed only deal with the synchronous subset only: `https://github.com/marcpouzet/zrun`. It builds on the works 1/ "A Coiterative Characterization of Synchronous Stream Functions", by Caspi and Pouzet, CMCS, 1998 (VERIMAG tech. report, 1997) and 2/ "The semantics and execution of a synchronous block-diagram language", by Edwards and Lee, Science of Computer Programming 2006.

7.6 Probabilistic Programming

Participants Guillaume Baudart, Marc Pouzet.

7.6.1 Reactive Probabilistic Programming

Synchronous languages were introduced to design and implement real-time embedded systems with a (justified) enphasis on determinacy. Yet, they interact with a physical environment that is only partially known and are implemented on architectures subject to failures and noise (e.g., channels, variable communication delays or computation time). Dealing with uncertainties is useful for online monitoring, learning, statistical testing or to build simplified models for faster simulation. Actual synchronous and languages provide limited support for modeling the non-deterministic behaviors that are omnipresent in embedded systems.

In collaboration with Louis Mandel (IBM), Erik Atkinson, Michael Carbin and Benjamin Sherman (MIT). We have designed ProbZelus, an extension of Zelus with probabilistic constructs to model uncertainties and perform inference-in-the-loop. The language makes it possible to describe probabilistic models in interaction with an observable environment. At runtime, a set of inference techniques can be used to learn the distributions of model parameters from observed data.

ProbZelus, is the first synchronous probabilistic programming language,combining language constructs for streams (reactivity) with those for probabilistic programming thus enablinginference-in-the-loop. We gave a measure-based co-iterative semantics for ProbZelus that forms the basis of a compiler and demonstrate a semantics-preserving compilation strategy to a first-order functional language: μF.

We defined the semantics of multiple inference algorithms on μF including particle filtering and delayed sampling: a semi-symbolic inference scheme. We then introduced a novel streaming delayed sampling implementation which enables partial exact inference over infinite streams in bounded memory for a large class of models.

ProbZelus is implemented on top of Zelus and available on Github[9]. The main article *Reactive Probabilistic Programming* was presented the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 2020) [1] and a short introduction to ProbZelus was presented at the Journées Francophones des Langages Applicatifs (JFLA 2020) [18].

7.6.2 Compiling Stan to Generative Probabilistic Languages

Stan is a probabilistic programming language that is popular in the statistics community, with a high-level syntax for expressing probabilistic models. Stan differs by nature from generative probabilistic programming languages like Church,Anglican, or Pyro. We proposed a comprehensive compilation scheme to compile any Stan model to a generative language and proved its correctness. We use our compilation scheme to build two new backends for the Stanc3 compiler targeting Pyro and NumPyro. Experimental results show that the NumPyro backend yields significant speedup compared to Stan on existing benchmarks.

[9] `https://github.com/IBM/probzelus`

Our compiler leverages the rich set of Pyro and Numpyro features for Stan users. Building on Pyro we thus extended Stan with support for explicit variational inference guides and deep probabilistic models, i.e., probabilistic models involving neural networks. The compiler is available on GitHub[10] and an article is currently under submission.

7.7 Identification of matrix operations for Compute-In-Memory architectures from a high-level Machine Learning framework

Participants Andi Drebes.

Compute-In-Memory (CIM) architectures are capable of performing certain performance-critical operations directly in memory (e.g., matrix multiplications) and represent a promising approach to partially eliminate the bottleneck of traditional von Neumann-based architectures resulting from long-distance communication between main memory and processing units.

In order for applications to benefit from such architectures, their operations must be divided into highly parallel, uniform operations eligible for in-memory computation and control logic that cannot benefit from CIM and that must be carried out by conventional computing devices. It is crucial for this process that as many eligible operations as possible are identified and effectively processed in memory, resulting only in as few computations as possible carried out on the conventional cores.

The programmability of CIM architectures is a key factor for its overall success. Manual identification of eligible operations and mapping to hardware resources is tedious, error-prone and requires detailed knowledge of the target architecture and therefore does not represent a viable approach to program CIM architectures.

With our partners from the MNEMOSENE project, we have developed a compilation toolchain that unburdens programmers from technical details of CIM architectures by allowing them to express algorithms at a high level of abstraction and that automates parallelization, orchestration and the mapping of operations to the CIM architecture. The solution integrates the Loop Tactics [39] declarative polyhedral pattern recognition and transformation framework into Tensor Comprehensions [38], a framework generating highly optimized kernels for accelerators from an abstract, mathematical notation for tensor operations. The compilation flow performs a set of dedicated optimizations aiming at enabling the reliable detection of computational patterns and their efficient mapping to CIM accelerators.

Early results were published at the 10th International Workshop on Polyhedral Compilation Techniques (IMPACT). In 2020, we extented this work for Multi-level Intermediate Representations and generalized our solution in a Tensor Comprehension front-end for the MLIR framework [32] and an multi-level IR optimizer. Follow-up work has led to an approach for progressive raising in multi-level intermediate representations

7.8 Progressive raising in Multi-Level Intermediate Representations

Participants Andi Drebes.

Multi-level intermediate representations (IR) are a promising approach for lowering the design costs for domain-specific compilers by providing a reusable, extensible and flexible framework for expressing domain-specific and high-level abstractions directly in the IR. However, while such frameworks support progressive lowering of high-level representations to low-level IR, they lack support for transformations in the opposite direction from low-level representations to higher ones. This means that the entry point into the compilation pipeline defines the highest level of abstraction for all subsequent transformations, limiting the set of applicable optimizations. General-purpose languages are particularly impacted by

[10]https://github.com/deepppl/stanc3

these limitations as their semantics are usually not rich enough to model the abstractions required by high-level transformations.

In collaboration with partners from TU Eindhoven, The University of Edinburgh and Google, we have developed an approach that allows compiler writers to declaratively specify transformations that raise low-level representations to high-level representations in multi-level IRs. The approach has been implemented on top of the MLIR framework [32] and is published as the *Multi-Level Tactics* Open Source project.[11] We have synthesized the concept of progressive raising, its implementation in Multi-Level Tactics and a demonstration of progressive raising from affine loop nests specified in a general-purpose language to high-level linear algebra operations with significant improvements of performance in a scientific paper, which was accepted for publication at the International Symposium on Code Generation and Optimization (CGO) 2021 [16].

8 Bilateral contracts and grants with industry

8.1 Bilateral contracts with industry

Collaboration with Airbus Our work on multi-clock Lustre programs is funded by a contract with Airbus.

9 Partnerships and cooperations

9.1 European initiatives

9.1.1 FP7 & H2020 Projects

TETRAMAX

Title: TEchnology TRAnsfer via Multinational Application eXperiments

Duration: 09/2017 - 12/2021

Coordinator: Rainer Leupers

Partners:

- AMG TECHNOLOGY OOD (Bulgaria)
- BUDAPESTI MUSZAKI ES GAZDASAGTUDOMANYI EGYETEM (Hungary)
- FUNDINGBOX ACCELERATOR SP ZOO (Poland)
- INSTITUT JOZEF STEFAN (Slovenia)
- RHEINISCH-WESTFAELISCHE TECHNISCHE HOCHSCHULE AACHEN (Germany)
- RUHR-UNIVERSITAET BOCHUM (Germany)
- SVEUCILISTE U ZAGREBU FAKULTET ELEKTROTEHNIKE I RACUNARSTVA (Croatia)
- TALLINNA TEHNIKAULIKOOL (Estonia)
- TAMPEREEN KORKEAKOULUSAATIO SR (Finland)
- TECHNISCHE UNIVERSITAET MUENCHEN (Germany)
- TECHNISCHE UNIVERSITEIT DELFT (Netherlands)
- THE UNIVERSITY OF EDINBURGH (UK)
- THINK SILICON RESEARCH AND TECHNOLOGY SINGLE MEMBER SA (Greece)
- UNIVERSITA DI PISA (Italy)

[11] https://github.com/LoopTactics/mlir

- UNIVERSITAT POLITECNICA DE CATALUNYA (Spain)
- UNIVERSITEIT GENT (Belgium)
- VSB - Technical University of Ostrava (Czech Republic)
- VYSOKE UCENI TECHNICKE V BRNE (Czech Republic)
- ZENIT ZENTRUM FUR INNOVATION UND TECHNIK IN NORDRHEIN-WESTFALEN GMBH (Germany)

Inria contact: Timothy Bourke

Summary: TETRAMAX is funded by the H2020 “Smart Anything Everywhere (SAE)” initiative. The overall ambition is to build and leverage a European Competence Center Network in customized low-energy computing, providing easy access for SMEs and mid-caps to novel CLEC technologies via local contact points. This is a bidirectional interaction: SMEs can demand CLEC technologies and solutions via the network, and vice versa academic research institutions can actively and effectively offer their new technologies to European industries. Furthermore, TETRAMAX wants to support 50+ industry clients and 3rd parties with innovative technologies, using different kinds of Technology Transfer Experiments (TTX) to accelerate innovation within European industries and to create a competitive advantage in the global economy.

MNEMOSENE

Title: Computation-in-memory architecture based on resistive devices

Duration: 1/2018 - 6/2021

Coordinator: Said Hamdioui

Partners:

- ARM LIMITED (UK)
- EIDGENOESSISCHE TECHNISCHE HOCHSCHULE ZUERICH (Switzerland)
- IBM RESEARCH GMBH (Switzerland)
- INTELLIGENTSIA CONSULTANTS SARL (Luxembourg)
- RHEINISCH-WESTFAELISCHE TECHNISCHE HOCHSCHULE AACHEN (Germany)
- STICHTING IMEC NEDERLAND (Netherlands)
- TECHNISCHE UNIVERSITEIT DELFT (Netherlands)
- TECHNISCHE UNIVERSITEIT EINDHOVEN (Netherlands)

Inria contact: Andi Drebes

Summary: MNEMOSENE aims at demonstrating a new computation-in-memory (CIM) computer architecture based on resistive devices, together with its required programming flow and interface. MNEMOSENE targets advanced explorative technology development at TRL 2 (technology concept formulation) and TRL3 (experimental proof-of-concept) and represents a first step towards the development of a fully operational CIM based computer, which MNEMOSENE consortium partners believe will require 9 to 12 years of further research after project completion.

EUROLAB4HPC2

Title: Consolidation of European Research Excellence in Exascale HPC Systems

Duration: 5/2018 - 4/2020

Coordinator: Per Stenström

Partners:

- BARCELONA SUPERCOMPUTING CENTER - CENTRO NACIONAL DE SUPERCOMPUTACION (Spain)
- ECOLE POLYTECHNIQUE FEDERALE DE LAUSANNE (Switzerland)
- EIDGENOESSISCHE TECHNISCHE HOCHSCHULE ZUERICH (Switzerland)
- IDRYMA TECHNOLOGIAS KAI EREVNAS (Greece)
- RHEINISCH-WESTFAELISCHE TECHNISCHE HOCHSCHULE AACHEN (Germany)
- THE UNIVERSITY OF EDINBURGH (UK)
- THE UNIVERSITY OF MANCHESTER (UK)
- UNIVERSITAET AUGSBURG (Germany)
- UNIVERSITAET STUTTGART (Germany)
- UNIVERSITEIT GENT (Belgium)

Inria contact: Albert Cohen

Summary: High-Performance Computing (HPC) systems are of vital importance to the progress of science and technology. Europe has made significant progress in becoming a leader in HPC through industrial and application providers. In addition ETP4HPC is driving a European HPC vision towards exascale systems. Despite such gains, excellence in HPC systems research is fragmented across Europe. Eurolab4HPC has the bold overall goal to strengthen academic research excellence and innovation in HPC in Europe.

9.2 National initiatives

9.2.1 ANR

The ANR JCJC project “FidelR” led by T. Bourke began in 2020 and continues for four years.

9.2.2 FUI: Fonds unique interministériel

Modeliscale contract (AAP-24) Using Modelica at scale to model and simulate very large Cyber-Physical Systems. Principal industrial partner: Dassault-Systèmes. INRIA contacts are Benoit Caillaud (HYCOMES, Rennes) and Marc Pouzet (PARKAS, Paris).

9.2.3 Programme d’Investissements d’Avenir (PIA)

ES3CAP collaborative project (Bpifrance) Develop a software and hardware platform for tomorrow’s intelligent systems. PARKAS collaborates with the industrial participants ANSYS/Esterel Technologies, Kalray, and Safran Electronics & Defense. Inria contacts are Marc Pouzet (PARKAS, Paris) and Fabrice Rastello (CORSE, Grenoble).

9.2.4 Others

Inria Project Lab (IPL) Modeliscale This project treats the modelling and analysis of Cyber-Physical Systems at large scale. The PARKAS team contributes their expertise in programming language design for reactive and hybrid systems to this multi-team effort.

10 Dissemination

10.1 Promoting scientific activities

10.1.1 Scientific events: organisation

Member of the organizing committees

- T. Bourke and M. Pouzet were coorganizers of the 27th International Open Workshop on Synchronous Programming (Synchron 2020).[12]

10.1.2 Scientific events: selection

Chair of conference program committees

- T. Bourke was program chair for the 19th ACM/IEEE International conference on Embedded Software (EMSOFT 2020).

Member of the conference program committees

- T. Bourke served on the program committee of the 32nd Euromicro Conference on Real-Time Systems (ECRTS 2020).
- T. Bourke served on the program committee of The American Modelica Conference 2020.
- T. Bourke served on the program committee of Asian Modelica Conference 2020.
- T. Bourke served on the program committee of the 23rd International Workshop on Software and Compilers for Embedded Systems (SCOPES 2020).
- M. Pouzet served on the program comittee of SCOPES 2020, EMSOFT 2020 and FDL 2020.
- G. Baudart served on the program committee of the Industry Track of the ACM International Conference on Distributed and Event-Based System (DEBS 2020).
- G. Baudart served on the program committee of the ACM SIGPLAN/SIGBED International Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES 2020)
- G. Baudart served on the program committee of the ACM/IEEE International conference on Embedded Software (EMSOFT 2020).
- G. Baudart served on the program committee of the Forum on specification & Design Languages (FDL 2020).

Reviewer

- T. Bourke was an external reviewer for the 29th European Symposium on Programming (ESOP 2020).
- T. Bourke was an external reviewer for the 23rd International Conference on Foundations of Software Science and Computation Structures (FoSSaCS 2020).
- T. Bourke was an external reviewer for the Haskell Symposium 2020.
- T. Bourke was an external reviewer for the 4th Workshop on Models for Formal Analysis of Real Systems (MARS 2020).

[12] https://synchron2020.inria.fr

10.1.3 Journal

Reviewer - reviewing activities

- T. Bourke reviewed articles for ACM Transactions on Cyber-Physical Systems.
- T. Bourke reviewed articles for the Journal of Logical and Algebraic Methods in Programming.
- T. Bourke reviewed articles for Science of Computer Programming.
- G. Baudart reviewed articles for the ACM Transactions on Embedded Computing Systems.
- G. Baudart reviewed articles for the IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems.

10.2 Teaching - Supervision - Juries

10.2.1 Teaching

- Marc Pouzet is Director of Studies for the CS department, at ENS.
- Licence : M. Pouzet & T. Bourke: "Operating Systems" (L3), Lectures and TDs, ENS, France.
- Master : M. Pouzet, G. Baudart, & T. Bourke, "Models and Languages for Reactive Systems" (M1), Lectures and TDs, ENS, France.
- Master: M. Pouzet & T. Bourke & G. Baudart: "Synchronous Systems" (M2), Lectures and TDs, MPRI, France
- Master: M. Pouzet: "Synchronous Reactive Languages" (M2), Lectures, Master COMASIC (École Polytechnique) and FIL (Université Paris-Sud, Saclay), France
- Master: T. Bourke: "A Programmer's introduction to Computer Architectures and Operating Systems" (M1), 32h, École Polytechnique, France
- Master: G. Baudart: "Synchronous Programming" (M2), TDs, Université de Paris, France
- Bachelor: T. Bourke: "A Programmer's introduction to Computer Architectures and Operating Systems" (L2), 32h, École Polytechnique, France
- Internships T. Bourke participated in reviewing the L3 and M1 internships of students at the ENS, France.

10.2.2 Supervision

- PhD: Lélio Brun, under review, supervised by T. Bourke and M. Pouzet, defended in June 2020.
- PhD in progress: Ismail Lakhim-Bennani, 2nd year, supervised by M. Pouzet, G. Frehse, and T. Bourke.
- PhD in progress: Paul Jeanmaire, 1st year, supervised by T. Bourke and M. Pouzet.
- PhD in progress: Baptiste Pauget, 1st year, supervised by M. Pouzet. CIFRE (ANSYS Toulouse and INRIA PARKAS).
- PhD in progress: Basile Pesin, 1st year, supervised by T. Bourke and M. Pouzet.

11 Scientific production

11.1 Major publications

[1] *Best Paper*
G. Baudart, L. Mandel, E. Atkinson, B. Sherman, M. Pouzet and M. Carbin. 'Reactive probabilistic programming'. In: *PLDI 2020 - 41th ACM SIGPLAN International Conference in Programming Language Design and Implementation*. London / Virtual, United Kingdom, June 2020. DOI: 10.1145/3385412.3386009. URL: https://hal.inria.fr/hal-03051954.

[2] T. Bourke, L. Brun, P.-E. Dagand, X. Leroy, M. Pouzet and L. Rieg. 'A Formally Verified Compiler for Lustre'. In: *PLDI 2017 - 38th ACM SIGPLAN Conference on Programming Language Design and Implementation*. ACM. Barcelone, Spain, June 2017. URL: https://hal.inria.fr/hal-01512286.

[3] T. Bourke, F. Carcenac, J.-L. Colaço, B. Pagano, C. Pasteur and M. Pouzet. 'A Synchronous Look at the Simulink Standard Library'. In: *EMSOFT 2017 - 17th International Conference on Embedded Software*. Seoul, South Korea: ACM Press, Oct. 2017, p. 23. URL: https://hal.inria.fr/hal-01575631.

[4] T. Bourke, J.-L. Colaço, B. Pagano, C. Pasteur and M. Pouzet. 'A Synchronous-based Code Generator For Explicit Hybrid Systems Languages'. In: *International Conference on Compiler Construction (CC)*. LNCS. London, United Kingdom, July 2015. URL: https://hal.inria.fr/hal-01242732.

[5] L. Gérard, A. Guatto, C. Pasteur and M. Pouzet. 'A modular memory optimization for synchronous data-flow languages: application to arrays in a lustre compiler'. In: *Proceedings of the 13th ACM SIGPLAN/SIGBED International Conference on Languages, Compilers, Tools and Theory for Embedded Systems*. Beijing, China: ACM, June 2012, pp. 51–60. DOI: 10.1145/2248418.2248426. URL: https://hal.inria.fr/hal-00728527.

[6] J. C. Juega, S. Verdoolaege, A. Cohen, J. I. Gómez, C. Tenllado and F. Catthoor. 'Patterns for parallel programming on GPUs'. In: *Patterns for parallel programming on GPUs*. Ed. by F. Magoulès. Vol. Evaluation of State-of-the-Art Parallelizing Compilers Generating CUDA Code for Heterogeneous CPU/GPU Computing. ISBN 978-1-874672-57-9. Saxe-Cobourg, 2013. URL: https://hal.archives-ouvertes.fr/hal-01257261.

[7] L. Mandel, F. Plateau and M. Pouzet. 'Static Scheduling of Latency Insensitive Designs with Lucy-n'. In: *FMCAD 2011 - Formal Methods in Computer Aided Design*. Austin, TX, United States, Oct. 2011. URL: https://hal.inria.fr/hal-00654843.

[8] R. Morisset, P. Pawan and F. Zappa Nardelli. 'Compiler testing via a theory of sound optimisations in the C11/C++11 memory model'. In: *PLDI 2013 - 34th ACM SIGPLAN conference on Programming language design and implementation*. Seattle, WA, United States: ACM, June 2013, pp. 187–196. DOI: 10.1145/2491956.2491967. URL: https://hal.inria.fr/hal-00909083.

[9] A. Pop and A. Cohen. 'OpenStream: Expressiveness and Data-Flow Compilation of OpenMP Streaming Programs'. In: *ACM Transactions on Architecture and Code Optimization* 9.4 (2013). Selected for presentation at the HiPEAC 2013 Conf. DOI: 10.1145/2400682.2400712. URL: https://hal.inria.fr/hal-00786675.

[10] J. Sevcik, V. Vafeiadis, F. Zappa Nardelli, S. Jagannathan and P. Sewell. 'CompCertTSO: A Verified Compiler for Relaxed-Memory Concurrency'. In: *Journal of the ACM (JACM)* 60.3 (2013), art. 22:1–50. DOI: 10.1145/2487241.2487248. URL: https://hal.inria.fr/hal-00909076.

[11] V. Vafeiadis, T. Balabonski, S. Chakraborty, R. Morisset and F. Zappa Nardelli. 'Common compiler optimisations are invalid in the C11 memory model and what we can do about it'. In: *POPL 2015 - 42nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. Mumbai, India, Jan. 2015. URL: https://hal.inria.fr/hal-01089047.

11.2 Publications of the year

International journals

[12] T. Bourke, L. Brun and M. Pouzet. 'Mechanized semantics and verified compilation for a dataflow synchronous language with reset'. In: *Proceedings of the ACM on Programming Languages* 4.POPL (22nd Jan. 2020), pp. 1–29. DOI: 10.1145/3371112. URL: https://hal.inria.fr/hal-02426573.

[13] A. Cohen and J. Zhao. 'Flextended Tiles: a Flexible Extension of Overlapped Tiles for Polyhedral Compilation'. In: *ACM Transactions on Architecture and Code Optimization* (1st Jan. 2020). DOI: 10.1145/3369382. URL: https://hal.inria.fr/hal-02458507.

International peer-reviewed conferences

[14] G. Baudart, L. Mandel, E. Atkinson, B. Sherman, M. Pouzet and M. Carbin. 'Reactive probabilistic programming'. In: PLDI 2020 - 41th ACM SIGPLAN International Conference in Programming Language Design and Implementation. London / Virtual, United Kingdom, 6th June 2020. DOI: 10.1145/3385412.3386009. URL: https://hal.inria.fr/hal-03051954.

[15] ***Best Paper***
A. Cha, E. Wittern, G. Baudart, J. C. Davis, L. Mandel and J. A. Laredo. 'A Principled Approach to GraphQL Query Cost Analysis'. In: ESEC/FSE 2020 - 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering. Sacramento / Virtual, United States: https://2020.esec-fse.org/, 6th Nov. 2020. DOI: 10.1145/3368089.3409670. URL: https://hal.inria.fr/hal-03117800.

[16] L. Chelini, A. Drebes, O. Zinenko, A. Cohen, N. Vasilache, T. Grosser and H. Corporaal. 'Progressive Raising in Multi-level IR'. In: CGO 2021 : International Symposium on Code Generation and Optimization. International Conference on Code Generation and Optimization (CGO). Seoul / Virtual, South Korea, 27th Feb. 2021. URL: https://hal.inria.fr/hal-03139764.

[17] L. Grimm, S. Smyth, A. Schulz-Rosengarten, R. von Hanxleden and M. Pouzet. 'From Lustre to Graphical Models and SCCharts'. In: FDL 2020 - Forum for Specification and Design Languages. 2020 Forum for Specification and Design Languages (FDL). Kiel, Germany: http://fdl-conference.org/, 3rd Nov. 2020. DOI: 10.1109/FDL50818.2020.9232944. URL: https://hal.inria.fr/hal-03128683.

Conferences without proceedings

[18] G. Baudart, L. Mandel, M. Pouzet, E. Atkinson, B. Sherman and M. Carbin. 'Programmation d'Applications Réactives Probabilistes'. In: JLFA 2020 - Journées Francophones des Langages Applicatifs. Gruissan, France: http://jfla.inria.fr/jfla2020.html, 29th Jan. 2020. URL: https://hal.inria.fr/hal-02430070.

[19] N. M. Nobre, A. Drebes, G. Riley and A. Pop. 'Bounded Stream Scheduling in Polyhedral OpenStream'. In: IMPACT 2020 - 10th International Workshop on Polyhedral Compilation Techniques. Bologna, Italy, 22nd Jan. 2020. URL: https://hal.inria.fr/hal-02441182.

Doctoral dissertations and habilitation theses

[20] L. Brun. 'Mechanized semantics and verified compilation for a dataflow synchronous language with reset'. Université Paris sciences et lettres, 6th July 2020. URL: https://tel.archives-ouvertes.fr/tel-03068862.

Reports & preprints

[21] G. Baudart, J. Burroni, M. Hirzel, L. Mandel and A. Shinnar. *Extending Stan for Deep Probabilistic Programming*. 21st Jan. 2021. URL: https://hal.inria.fr/hal-03117782.

[22] G. Iooss, M. Pouzet, A. Cohen, D. Potop-Butucaru, J. Souyris, V. Bregeon and P. Baufreton. *1-Synchronous Programming of Large Scale, Multi-Periodic Real-Time Applications with Functional Degrees of Freedom*. 12th Mar. 2020. URL: https://hal.inria.fr/hal-02495471.

Other scientific publications

[23] A. Drebes, L. Chelini, O. Zinenko, A. Cohen, H. Corporaal, T. Grosser, K. Vadivel and N. Vasilache. *TC-CIM: Empowering Tensor Comprehensions for Computing-In-Memory*. Bologna, Italy, 22nd Jan. 2020. URL: https://hal.inria.fr/hal-02441163.

11.3 Cited publications

[24] S. Blazy, Z. Dargaye and X. Leroy. 'Formal Verification of a C Compiler Front-End'. In: *FM 2006: Int. Symp. on Formal Methods*. Vol. 4085. Lecture Notes in Computer Science. Springer-Verlag, 2006, pp. 460–475. URL: http://gallium.inria.fr/~xleroy/publi/cfront.pdf.

[25] P. Caspi, J.-L. Colaço, L. Gérard, M. Pouzet and P. Raymond. 'Synchronous Objects with Scheduling Policies: Introducing safe shared memory in Lustre'. In: *ACM International Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES)*. Dublin, June 2009.

[26] A. Cohen, M. Duranton, C. Eisenbeis, C. Pagetti, F. Plateau and M. Pouzet. '*N*-Synchronous Kahn Networks: a Relaxed Model of Synchrony for Real-Time Systems'. In: *ACM International Conference on Principles of Programming Languages (POPL'06)*. Charleston, South Carolina, USA, Jan. 2006.

[27] A. Cohen, L. Mandel, F. Plateau and M. Pouzet. 'Abstraction of Clocks in Synchronous Data-flow Systems'. In: *The Sixth ASIAN Symposium on Programming Languages and Systems (APLAS)*. Bangalore, India, Dec. 2008.

[28] J.-L. Colaço, B. Pagano and M. Pouzet. 'A Conservative Extension of Synchronous Data-flow with State Machines'. In: *ACM International Conference on Embedded Software (EMSOFT'05)*. Jersey city, New Jersey, USA, Sept. 2005.

[29] *The Coq proof Assistant*. http://coq.inria.fr. 2019.

[30] J. Forget. 'Un Langage Synchrone pour les Systèmes Embarqués Critiques Soumis à des Contraintes Temps Réel Multiples'. PhD thesis. Université de Toulouse, Nov. 2009.

[31] J. Forget, F. Boniol, D. Lesens and C. Pagetti. 'A Real-Time Architecture Design Language for Multi-Rate Embedded Control Systems'. In: *proceedings 25th ACM symposium on Applied Computing (SAC'10)*. Ed. by S. Y. Shin, S. Ossowski, M. Schumacher, M. J. Palakal and C.-C. Hung. Sierre, Switzerland, Mar. 2010, pp. 527–534.

[32] C. Lattner, M. Amini, U. Bondhugula, A. Cohen, A. Davis, J. A. Pienaar, R. Riddle, T. Shpeisman, N. Vasilache and O. Zinenko. 'MLIR: Scaling Compiler Infrastructure for Domain Specific Computation'. In: *International Symposium on Code Generation and Optimization (CGO)*. Feb. 2021.

[33] X. Leroy. *The Compcert verified compiler*. 2009. URL: http://compcert.inria.fr/doc/index.html.

[34] L. Mandel, F. Plateau and M. Pouzet. 'Lucy-n: a n-Synchronous Extension of Lustre'. In: *Tenth International Conference on Mathematics of Program Construction (MPC 2010)*. Québec, Canada, June 2010. URL: http://www.lri.fr/~mandel/papiers/MandelPlateauPouzet-MPC-10.pdf.

[35] L. Mandel, F. Plateau and M. Pouzet. 'Static Scheduling of Latency Insensitive Designs with Lucy-n'. In: *International Conference on Formal Methods in Computer-Aided Design (FMCAD)*. Austin, Texas, USA, Oct. 2011.

[36] F. Plateau. 'Modèle n-synchrone pour la programmation de réseaux de Kahn à mémoire bornée'. PhD thesis. Orsay, France: Université Paris-Sud 11, June 2010. URL: https://www.lri.fr/~mandel/lucy-n/~plateau/these/.

[37] M. Pouzet and P. Raymond. 'Modular Static Scheduling of Synchronous Data-flow Networks: An efficient symbolic representation'. In: *ACM International Conference on Embedded Software (EMSOFT'09)*. Grenoble, France, Oct. 2009.

[38] N. Vasilache, O. Zinenko, T. Theodoridis, P. Goyal, Z. Devito, W. S. Moses, S. Verdoolaege, A. Adams and A. Cohen. 'The Next 700 Accelerated Layers: From Mathematical Expressions of Network Computation Graphs to Accelerated GPU Kernels, Automatically'. In: *ACM Transactions on Architecture and Code Optimization (TACO)* 16.4 (Oct. 2019), Article 38. DOI: 10.1145/3355606.

[39] O. Zinenko, L. Chelini and T. Grosser. *Declarative Transformations in the Polyhedral Model*. Research Report RR-9243. Inria ; ENS Paris - Ecole Normale Supérieure de Paris ; ETH Zurich ; TU Delft ; IBM Zürich, Dec. 2018. URL: https://hal.inria.fr/hal-01965599.