IN PARTNERSHIP WITH:
CNRS

Ecole normale supérieure de Paris

Activity Report 2019

Project-Team PARKAS

Parallélisme de Kahn Synchrone

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

RESEARCH CENTER
Paris

THEME
Embedded and Real-time Systems

Table of contents

1. Team, Visitors, External Collaborators **1**
2. Overall Objectives **2**
3. Research Program **2**
3.1. Programming Languages for Cyber-Physical Systems 2
3.2. Efficient Compilation for Parallel and Distributed Computing 3
3.3. Validation and Proof of Compilers 4
3.3.1. Lustre: 4
3.3.2. C/C++: 4
4. Application Domains **5**
4.1. Embedded Control Software 5
4.2. Hybrid Systems Design and Simulation 5
5. New Software and Platforms **5**
5.1. Cmmtest 5
5.2. GCC 5
5.3. Heptagon 6
5.4. isl 6
5.5. Lem 6
5.6. Lucid Synchrone 7
5.7. Lucy-n 7
5.8. Ott 7
5.9. PPCG 8
5.10. ReactiveML 8
5.11. SundialsML 8
5.12. Zelus 9
5.13. Telamon 9
5.14. Vélus 9
5.15. Tensor Comprehensions 10
5.16. Aftermath 10
5.17. MPPcodegen 10
5.18. MPP 10
5.19. Obelisk 10
6. New Results **11**
6.1. Efficiently Subtyping Union Types 11
6.2. Fast and reliable unwinding via DWARF tables 11
6.3. Verified compilation of Lustre 11
6.3.1. Compiling the modular reset construct. 12
6.3.2. Non-normalized Lustre. 12
6.3.3. Strengthening the correctness theorem. 12
6.4. Specifying multi-clock Lustre programs 12
6.4.1. Harmonic clocks 13
6.4.2. New Intermediate Language MObc (Multi Object Code) 13
6.4.3. Clocking constraints, communication latencies, and constraint solving 13
6.5. The Zelus Language 13
6.5.1. Compiler Internals: Static Typing and Compiler Organisation 14
6.5.2. Co-simulation as Function Lifting 14
6.5.3. QSS-based Simulation 14
6.5.4. Property Based Testing of Hybrid Programs 15
6.6. Reactive Probabilistic Programming 15

6.7. Identification of matrix operations for Compute-In-Memory architectures from a high-level Machine Learning framework 15
6.8. Applying reinforcement learning to improve a branch-and-bound optimizing compiler 16
7. Bilateral Contracts and Grants with Industry . . . 16
7.1. Bilateral Contracts with Industry 16
7.2. Bilateral Grants with Industry 17
8. Partnerships and Cooperations . . . 17
8.1. National Initiatives 17
8.1.1. ANR 17
8.1.2. FUI: Fonds unique interministériel 17
8.1.3. Programme d'Investissements d'Avenir (PIA) 17
8.1.4. Others 17
8.2. European Initiatives 17
8.3. International Initiatives 18
9. Dissemination . . . 18
9.1. Promoting Scientific Activities 18
9.1.1. Scientific Events: Selection 18
9.1.1.1. Chair of Conference Program Committees 18
9.1.1.2. Member of the Conference Program Committees 18
9.1.1.3. Reviewer 18
9.1.2. Journal 19
9.1.3. Research Administration 19
9.2. Teaching - Supervision - Juries 19
9.2.1. Teaching 19
9.2.2. Supervision 19
9.2.3. Juries 19
9.3. Popularization 19
10. Bibliography . . . 20

Project-Team PARKAS

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

Keywords:

Computer Science and Digital Science:

A1.1.1. - Multicore, Manycore
A1.1.3. - Memory models
A2.1.1. - Semantics of programming languages
A2.1.4. - Functional programming
A2.1.6. - Concurrent programming
A2.1.9. - Synchronous languages
A2.2.2. - Memory models
A2.2.4. - Parallel architectures
A2.2.5. - Run-time systems
A2.2.6. - GPGPU, FPGA...
A2.2.7. - Adaptive compilation
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
A3.1.3. - Distributed data
A6.2.1. - Numerical analysis of PDE and ODE
A6.2.7. - High performance computing

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
B9.5.1. - Computer science
B9.5.2. - Mathematics

1. Team, Visitors, External Collaborators

Research Scientists

Timothy Bourke [Inria, Researcher]
Francesco Zappa Nardelli [Inria, Senior Researcher, HDR]

Faculty Member

Marc Pouzet [Team leader, Univ Pierre et Marie Curie, Professor]

Post-Doctoral Fellow

Guillaume Iooss [Inria]

PhD Students

Ulysse Beaugnon [Ecole Normale Supérieure Paris, until Apr 2019]
Lelio Brun [Ecole Normale Supérieure Paris]
Basile Clément [Inria, until Nov 2019]
Ismail Lahkim Bennani [Inria]

Technical staff

Andi Drebes [Inria, Engineer]
Sophie Kaleba [Inria, Engineer, from Mar 2019 until Aug 2019]
Chandan Reddy Gopal [Inria, Engineer, until Aug 2019]
Adilla Susungi [Inria, Engineer, until Mar 2019]
Nicolas Tollenaere [Inria, Engineer, until Mar 2019]

Interns and Apprentices

Paul Jeanmaire [Inria, from Apr 2019 until Aug 2019]
Baptiste Pauget [Inria, from Apr 2019 until Aug 2019]
Remi Oudin [Inria, until Mar 2019]

External Collaborator

Paul Feautrier [Univ de Lyon]

2. Overall Objectives

2.1. 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 running on weak-memory processors.

To realize our research program, we develop languages (LUCID SYNCHRONE, REACTIVEML, LUCY-N, ZELUS), compilers (PPCG), contributions to open-source projects (isl, LLVM, gcc, SundialsML), tools to study language semantics (Ott) and to test optimization compilers in the presence of threads (cmmtest), and formalizations in Interactive Theorem Provers of language semantics (Vélus, *n*-synchrony, quasi-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 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.

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. Efficient Compilation for Parallel and Distributed Computing

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.

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

- Parallelism is about generating code capable of efficiently exploiting multiprocessors. Typically this amounts to maing (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 executed code and source programs. We develop techniques to formally prove the correctness of compilation passes for synchronous languages (Lustre), and to validate compilation optimization for C code in the presence of threads.

3.3.1. Lustre:

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.

3.3.2. C/C++:

The recently approved C11 and C++11 standards define a concurrency model for the C and C++ languages, which were originally designed without concurrency support. Their intent is to permit most compiler and hardware optimizations, while providing escape mechanisms for writing portable, high-performance, low-level code. Mainstream compilers are being modified to support the new standards. A subtle class of compiler bugs is the so-called concurrency compiler bugs, where compilers generate correct sequential code but break the concurrency memory model of the programming language. Such bugs are observable only when the miscompiled functions interact with concurrent contexts, making them particularly hard to detect. All previous techniques to test compiler correctness miss concurrency compiler bugs.

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. New Software and Platforms

5.1. Cmmtest

FUNCTIONAL DESCRIPTION: Cmmtest is a tool for hunting concurrency compiler bugs. The Cmmtest tool performs random testing of C and C++ compilers against the C11/C++11 memory model. A test case is any well-defined, sequential C program, for each test case, cmmtest:

compiles the program using the compiler and compiler optimisations that are being tested,

runs the compiled program in an instrumented execution environment that logs all memory accesses to global variables and synchronisations,

compares the recorded trace with a reference trace for the same program, checking if the recorded trace can be obtained from the reference trace by valid eliminations, reorderings and introductions.

Cmmtest identified several mistaken write introductions and other unexpected behaviours in the latest release of the gcc compiler. These have been promptly fixed by the gcc developers.

- Participants: Anirudh Kumar, Francesco Zappa Nardelli, Pankaj More, Pankaj Pawan, Pankaj Prateek Kewalramani and Robin Morisset
- Contact: Francesco Zappa Nardelli
- URL: http://www.di.ens.fr/~zappa/projects/cmmtest/

5.2. GCC

KEYWORDS: Compilation - Polyhedral compilation

FUNCTIONAL DESCRIPTION: The GNU Compiler Collection includes front ends for C, C++, Objective-C, Fortran, Java, Ada, and Go, as well as libraries for these languages (libstdc++, libgcj,...). GCC was originally written as the compiler for the GNU operating system. The GNU system was developed to be 100% free software, free in the sense that it respects the user's freedom.

- Participants: Albert Cohen, Feng Li, Nhat Minh Le, Riyadh Baghdadi and Tobias Grosser
- Contact: Albert Cohen
- URL: http://gcc.gnu.org/

5.3. 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.

- Participants: Adrien Guatto, Brice Gelineau, Cédric Pasteur, Eric Rutten, Gwenaël Delaval, Léonard Gérard and Marc Pouzet
- Partners: UGA - ENS Paris - Inria - LIG
- Contact: Gwenaël Delaval
- URL: http://heptagon.gforge.inria.fr

5.4. isl

FUNCTIONAL DESCRIPTION: isl is a library for manipulating sets and relations of integer points bounded by linear constraints. Supported operations on sets include intersection, union, set difference, emptiness check, convex hull, (integer) affine hull, integer projection, transitive closure (and over-approximation), computing the lexicographic minimum using parametric integer programming. It includes an ILP solver based on generalized basis reduction, and a new polyhedral code generator. isl also supports affine transformations for polyhedral compilation, and increasingly abstract representations to model source and intermediate code in a polyhedral framework.

- Participants: Albert Cohen, Sven Verdoolaege and Tobias Grosser
- Contact: Sven Verdoolaege
- URL: http://freshmeat.net/projects/isl

5.5. Lem

lightweight executable mathematics

FUNCTIONAL DESCRIPTION: Lem is a lightweight tool for writing, managing, and publishing large scale semantic definitions. It is also intended as an intermediate language for generating definitions from domain-specific tools, and for porting definitions between interactive theorem proving systems (such as Coq, HOL4, and Isabelle). As such it is a complementary tool to Ott. Lem resembles a pure subset of Objective Caml, supporting typical functional programming constructs, including top-level parametric polymorphism, datatypes, records, higher-order functions, and pattern matching. It also supports common logical mechanisms including list and set comprehensions, universal and existential quantifiers, and inductively defined relations. From this, Lem generates OCaml, HOL4, Coq, and Isabelle code.

- Participants: Francesco Zappa Nardelli, Peter Sewell and Scott Owens
- Contact: Francesco Zappa Nardelli
- URL: http://www.cl.cam.ac.uk/~pes20/lem/

5.6. Lucid Synchrone

FUNCTIONAL DESCRIPTION: Lucid Synchrone is a language for the implementation of reactive systems. It is based on the synchronous model of time as provided by Lustre combined with features from ML languages. It provides powerful extensions such as type and clock inference, type-based causality and initialization analysis and allows to arbitrarily mix data-flow systems and hierarchical automata or flows and valued signals.

RELEASE FUNCTIONAL DESCRIPTION: The language is still used for teaching and in our research but we do not develop it anymore. Nonetheless, we have integrated several features from Lucid Synchrone in new research prototypes described below. The Heptagon language and compiler are a direct descendent of it. The new language Zélus for hybrid systems modeling borrows many features originaly introduced in Lucid Synchrone.

- Contact: Marc Pouzet
- URL: http://www.di.ens.fr/~pouzet/lucid-synchrone/

5.7. Lucy-n

Lucy-n: an n-synchronous data-flow programming language

FUNCTIONAL DESCRIPTION: Lucy-n is a language to program in the n-synchronous model. The language is similar to Lustre with a buffer construct. The Lucy-n compiler ensures that programs can be executed in bounded memory and automatically computes buffer sizes. Hence this language allows to program Kahn networks, the compiler being able to statically compute bounds for all FIFOs in the program.

- Participants: Adrien Guatto, Albert Cohen, Louis Mandel and Marc Pouzet
- Contact: Albert Cohen
- URL: https://www.lri.fr/~mandel/lucy-n/

5.8. Ott

FUNCTIONAL DESCRIPTION: Ott is a tool for writing definitions of programming languages and calculi. It takes as input a definition of a language syntax and semantics, in a concise and readable ASCII notation that is close to what one would write in informal mathematics. It generates output:

a LaTeX source file that defines commands to build a typeset version of the definition,

a Coq version of the definition,

an Isabelle version of the definition, and

a HOL version of the definition.

Additionally, it can be run as a filter, taking a LaTeX/Coq/Isabelle/HOL source file with embedded (symbolic) terms of the defined language, parsing them and replacing them by typeset terms.

The main goal of the Ott tool is to support work on large programming language definitions, where the scale makes it hard to keep a definition internally consistent, and to keep a tight correspondence between a definition and implementations. We also wish to ease rapid prototyping work with smaller calculi, and to make it easier to exchange definitions and definition fragments between groups. The theorem-prover backends should enable a smooth transition between use of informal and formal mathematics.

- Participants: Francesco Zappa Nardelli, Peter Sewell and Scott Owens
- Contact: Francesco Zappa Nardelli
- URL: http://www.cl.cam.ac.uk/~pes20/ott/

5.9. PPCG

FUNCTIONAL DESCRIPTION: PPCG is our source-to-source research tool for automatic parallelization in the polyhedral model. It serves as a test bed for many compilation algorithms and heuristics published by our group, and is currently the best automatic parallelizer for CUDA and OpenCL (on the Polybench suite).

- Participants: Albert Cohen, Riyadh Baghdadi, Sven Verdoolaege and Tobias Grosser
- Contact: Sven Verdoolaege
- URL: http://freshmeat.net/projects/ppcg

5.10. ReactiveML

FUNCTIONAL DESCRIPTION: ReactiveML is a programming language dedicated to the implementation of interactive systems as found in graphical user interfaces, video games or simulation problems. ReactiveML is based on the synchronous reactive model due to Boussinot, embedded in an ML language (OCaml).

The Synchronous reactive model provides synchronous parallel composition and dynamic features like the dynamic creation of processes. In ReactiveML, the reactive model is integrated at the language level (not as a library) which leads to a safer and a more natural programming paradigm.

- Participants: Cédric Pasteur, Guillaume Baudart and Louis Mandel
- Contact: Guillaume Baudart

5.11. SundialsML

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 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 FUNCTIONAL DESCRIPTION: Adds support for v3.1.x of the Sundials Suite of numerical solvers.

Notably this release adds support for the new generic matrix and linear solver interfaces. The OCaml interface changes but the library is backward compatible with Sundials 2.7.0.

OCaml 4.02.3 or greater is now required and optionally OCamlMPI 1.03.

* New Sundials.Matrix and Sundials.LinearSolver modules. * Better treatment of integer type used for matrix indexing. * Refactor Dls and Sls modules into Sundials.Matrix. * Add confidence intervals to performance graph. * Miscellaneous improvements to configure script. * Potential incompatibility: changes to some label names: comm_fn -> comm, iter_type -> iter. * Untangle the ARKODE mass-solver interface from the Jacobian interface.

- Participants: Jun Inoue, Marc Pouzet and Timothy Bourke
- Partner: UPMC
- Contact: Marc Pouzet
- URL: http://inria-parkas.github.io/sundialsml/

5.12. Zelus

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.

- Participants: Marc Pouzet and Timothy Bourke
- Contact: Marc Pouzet

5.13. Telamon

KEYWORDS: Compilation - Monte-Carlo methods - Constraint Programming - GPU - Dense linear algebra

FUNCTIONAL DESCRIPTION: Telamon is a framework for the optimization of computational kernels for GPUs through efficient search in a well-behaved optimization space in which optimization decisions commute and satisfaction of constraints restricting legal optimizations.

- Contact: Ulysse Beaugnon
- URL: https://github.com/ulysseB/telamon/

5.14. Vélus

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

- Contact: Timothy Bourke
- URL: https://velus.inria.fr

5.15. Tensor Comprehensions

KEYWORDS: Machine learning - Matrix calculation - Polyhedral compilation - GPU - CUDA

FUNCTIONAL DESCRIPTION: Tensor Comprehensions (TC) is a notation based on generalized Einstein notation for computing on multi-dimensional arrays. TC greatly simplifies ML framework implementations by providing a concise and powerful syntax which can be efficiently translated to high-performance computation kernels, automatically.

RELEASE FUNCTIONAL DESCRIPTION: Integration of the loop tactics matching framework for identifying linear algebra operations and optimizing them.

- Partner: Eindhoven University of Technology
- Contact: Albert Cohen

5.16. Aftermath

KEYWORDS: Performance analysis - Runtime system - Parallel programming - High-performance calculation - Execution trace

FUNCTIONAL DESCRIPTION: Aftermath is a toolkit for building custom graphical tools for trace-based performance analysis of parallel programs, run-time systems and compilers.

- Partner: The University of Manchester
- Contact: Andi Drebes

5.17. 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/

5.18. MPP

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.

- Contact: Guillaume Iooss
- URL: https://github.com/guillaumeiooss/MPP

5.19. Obelisk

KEYWORDS: LaTeX - HTML - Ocaml

FUNCTIONAL DESCRIPTION: Obelisk is a simple tool which produces pretty-printed output from a Menhir parser file (.mly).

It is inspired from yacc2latex and it is also written in OCaml, but it is aimed at supporting features from Menhir instead of only those of ocamlyacc.

- Contact: Lelio Brun
- URL: https://github.com/Lelio-Brun/Obelisk

6. New Results

6.1. Efficiently Subtyping Union Types

Participant: Francesco Zappa Nardelli.

Julia is a programming language recently designed at MIT to support the needs of the scientific community. Julia occupies a unique position in the design landscape, it is a dynamic language with no type system, yet it has a surprisingly rich set of types and type annotations used to specify multimethod dispatch. The types that can be expressed in function signatures include parametric union types, covariant tuple types, parametric user-defined types with single inheritance, invariant type application, and finally types and values can be reified to appear in signatures. In 2017 with Vitek we started a research project to study the design and the pragmatic use of the Julia language, and formalised the Julia subtyping algorithm. In 2018 we have pursued this study, and we have proved correct the clever and space efficient algorithm relied upon by the Julia runtime. This has been published in [17].

6.2. Fast and reliable unwinding via DWARF tables

Participants: Theophile Bastian, Rémy Oudin, Francesco Zappa Nardelli.

DWARF is a widely-used debugging data format. DWARF is obviously relied upon by debuggers, but it plays an unexpected role in the runtime of high-level programming languages and in the implementation of program analysis tools. The debug information itself can be pervaded by subtle bugs, making the whole infrastructure unreliable. In this project we are investigating techniques and tools to perform validation and synthesis of the DWARF stack unwinding tables, to speedup DWARF-based unwinding, as well as exploring adventurous projects that can be built on top of reliable DWARF information.

We have built a tool that can validate DWARF unwind tables generated by mainstream compilers; the approach is effective, we found a problem in Clang table generation and several in GLIBC inline-assembly snippets. We also designed and implemented a tool that can synthesise DWARF unwind tables from binary that lacks them (e.g. because the compiler did not generate them - immediate applications: JITs assembly, inline assembly, ...). Additionally we have designed and implemented a ahead-of-time compiler of DWARF unwind tables to assembly, and an ad-hoc unwinder integrated with the defacto standard unwinder libuwind. It can speed up unwinding by a factor between 12x and 25x (depending on application), with a 2.5x size overhead for unwind information.

This work has been published in [13].

6.3. Verified compilation of Lustre

Participants: Timothy Bourke, Lélio Brun, Paul Jeanmaire, Marc Pouzet.

Vélus is a compiler for a subset of LUSTRE and SCADE that is specified in the Coq [28] Interactive Theorem Prover (ITP). It integrates the CompCert C compiler [34], [29] 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 MBD;
- 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.

This year we created a website for the project (https://velus.inria.fr) and made an initial release under an Inria non-commerical license (https://github.com/Inria/velus). T. Bourke's JCJC ("Jeune Chercheuse Jeune Chercheur") project *FidelR* was accepted for funding by the ANR: it aims to develop ITP-based techniques for treating state machines and interactive program verification. We also made progress on the compilation of the modular reset construct, the treatment of (non-normalized) Lustre, and our longer term goal of strengthening the main correctness theorem. These results are detailed below.

6.3.1. Compiling the modular reset construct.

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). Last year we showed how to encode the semantics of this construct in Coq. This year we focused on its compilation and the associated proof of correctness. We designed and implemented a new intermediate language that exposes different *step* and *reset* actions on node instances. This language facilitates the optimization of conditional statements in the generated code and permits the transformation to imperative code and its proof of correctness to be treated in two steps: one to introduce named memories and another to fix the sequential order of execution. This work forms the core of L. Brun's thesis, to be defended early next year, and an article accepted at the ACM SIGBED international conference on Principles of Programming Languages (POPL 2020).

6.3.2. Non-normalized Lustre.

Our previous work has focused on a subset of "normalized" programs where the form of expressions and equations is constrained to facilitate the compilation. We have generalized the definitions of syntax and semantics in our prototype compiler to accept non-normalized programs. This included simplifying and generalizing the formalization of clocks presented in [20]. With P. Jeanmaire (M2 internship), we have implemented a compilation pass to translate normalized programs from one syntactical form to another. The main challenge was to formally prove an alignment property (signals are present iff their clocks are true) that had been assumed until now. The proof is finished except for the inductive case for the reset construct which we hope to complete soon.

6.3.3. Strengthening the correctness theorem.

The current correctness theorem assumes that an accepted program can be given a semantics in terms of the mechanized model. It should be possible to prove this fact for programs that pass the initial type-checking and clock-checking algorithms, that can be scheduled, and which never invoke an undefined operation (such as a division by zero). We made good progress on this problem by defining an interpreter for normalized Lustre programs and showing that the results it calculates satisfy the semantic predicates. This initial work gives some useful insights into how to proceed. We presented it at the Synchron 2019 workshop.

6.4. Specifying multi-clock Lustre programs

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

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.

6.4.1. Harmonic clocks

We studied the extension of a synchronous language with periodic harmonic clocks based on the work of Mandel et al. [31], [37], [32], [35], [36] on n-synchrony and the extension proposed by Forget et al. [33]

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 ineternship, 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.

6.4.2. 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. [30] 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. [38] on the calculation of input/output relations to merge calculations. We are preparing and article on this subject.

6.4.3. Clocking constraints, communication latencies, and constraint solving

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 `fby` 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 has been prototyping these ideas in the academic Heptagon compiler.

This work is funded by a direct industrial contract with Airbus. In collaboration (this year) with Michel Angot Vincent Bregeon Jean Souyris (Airbus, R&D) and Matthieu Boitrel (Airbus BE).

6.5. The Zelus Language

Participants: Timothy Bourke, 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.

This year, we have pursued our work on the design, semantics and implementation of hybrid modeling language, in particular the treatment of Differential Algebraic Equations (DAEs) [23].

6.5.1. Compiler Internals: Static Typing and Compiler Organisation

The distribution with manual and examples is distributed at http://zelus.di.ens.fr (only Version 1, in binary form). Version 2 (the current active branch) is available in source form on Inria GitLab https://gitlab.inria.fr/parkas/zelus, on simple demand.

Several new experimentations have been done this year, in particular on the type system and an extensive rewritting of some compilation internals to simplify the code and make the generated code more shorter (in size) and more efficient.

6.5.2. Co-simulation as Function Lifting

Hybrid models in Zelus (that is, programs that mix discrete and continuous-time signals) are simulated using a single ODE and zero-crossing solvers only. All hybrid modeling languages (e.g., Simulink, Modelica, Ptolemy) act the same way, at least, single solver simulation is the default mechanism.

Its weaknesses are well known: any change of the dynamic, even local, calls for a global reset of the solver, making it slower for that later steps; the mix of a slow and fast signals slows down the whole simulation. Co-simulation is about running several solvers (or instances of the same) at the same time.

We proposed a limited (but useful) manner, by proving a way to internalize the solver to obtain, from a continuous-time function, a synchronous stream function. A preliminary experiment done this year was surprisingly and pleasingly simple to implement in Zelus. It consisted in defining a (higher-order) function *solve* that, given a continuous-time function *f* returns a stream function *solve f*. Given an input stream *x* and an increasing stream of time horizons *h*, *solve f(x, t)* returns the stream of approximated values. This function internalizes the ODE solver and the zero-crossing detection mechanism. The overall model is then a purely discrete-time, synchronous model. In particular, classical synchronization protocols between solvers can be programmed in the language itself, hence benefiting from the static checks that track typing, causality and initialization errors, properties that would be more difficult to ensure if programmed directly in C, for example. We think that it is even possible to write a formal synchronous specification of the simulation engine itself, that is, to program the function solve directly in Zelus. This experiment on co-simulation gives new insight on the semantics based on non standard analysis that we proposed and, more interestingly, to relate it to the proven and more classical semantics based on super-dense time studied and exploited by Edward Lee.

6.5.3. QSS-based Simulation

Quantized State Systems simulation (QSS) was introduced in the early 2000's by F. Cellier and E. Kofman as an alternative to time-based simulation, which is the dominant approach to ODE/DAE systems simulation.

Rather than linking QSS to Discrete Event Simulation, we have made a preliminary experiment to relate it to Synchronous Programming and its continuous time extension Zelus. Zelus is used to give a formal description of the QSS method that can be executed. We have described the very basic scheme called QSS (or QSS1) for which we can give a Zelus (hence executable) specification. Higher order schemes QSS2, 3, etc. can also be given an Zelus specification. Implicit schemes were also proposed by Kofman for a better handling of stiff systems. Higher order versions of BQSS are nontrivial; they are called LIQSS1, 2, 3, etc. and they can also be specified in Zelus. This preliminary work is done in collaboration with Albert Benveniste (Inria Hycomes, Rennes) and funded by the Modeliscale FUI project.

6.5.4. 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. To compute the gradients, we use Automatic Differentiation (AD) of the system under test and its specification. Together with François Bidet, we ported the well-known FADBAD++ library for AD written by Ole Stauning in 1997 to OCaml—the target language of Zélus. Our port is called FADBADml and is now released under an Inria license [4] and is available on opam.

6.6. Reactive Probabilistic Programming

Participant: Marc Pouzet.

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 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 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 2019, we started a new topic on *reactive probabilistic programming* under the initiative of Guillaume Baudart and Louis Mandel (IBM Research, Watson); in collaboration with Erik Atkinson, Michael Carbin and Benjamin Sherman (MIT). We have designed ProbZelus, an extension of Zelus with probabilistic programming constructs. 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. The main results are (1) the design of the ProbZelus compiler, the formalization of the static and dynamic semantics of the language that mixes deterministic and probabilistic components, (2) the design and implementation of inference methods which can be executed with bounded resources, (3) the evaluation of ProbZelus on a set of examples and case studies.

For the moment, ProbZelus is mainly a library of Zelus, with minor changes of the language itself (essentially the type system) [5] It exploits heavilly the higher-order nature of Zelus. E.g., if `f` is a stream function (with type `f: 'a -D-> 'b`) and `x` is a stream (with type `'a`), `Particule.infer f x: 'a -D-> 'b Distribution.t` implements an inference algorithm which computes a distribution for the result of type `'b` with a particule filter algorithm.

A prelimary report describes a part of this work [24] and a presentation at JFLA will be given in January 2020. ProbZelus is available in open source at https://github.com/IBM/probzelus since december 2019. Our purpose is to go beyond the library approach with a closer integration of probabilistic constructs and reactive constructs, with dedicated static analyses and compilation techniques to give static guaranties on the result of inference, efficient inference techniques tuned for reactive applications and that ensure execution in bounded time and space; efficient dedicated compilation techniques for probabilistic programs. Finally, the treatment of both discrete-time and continuous-time signals and systems must be investigated (only discrete-time is considered at the moment).

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

Participant: Andi Drebes.

[4] https://fadbadml-dev.github.io/FADBADml/

[5] Yet, higher-order was only used occasionally since then; hence an important implementation effort has been spent this year to make it work well.

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 [40] declarative polyhedral pattern recognition and transformation framework into Tensor Comprehensions [39], 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.

The results of this work have been submitted to the 10th International Workshop on Polyhedral Compilation Techniques (IMPACT).

6.8. Applying reinforcement learning to improve a branch-and-bound optimizing compiler

Participant: Basile Clement.

Frameworks for image processing, deep learning, etc., work with Directed Acyclic Graphs (DAGs) of computational operators that wrap high-performance libraries. The production of highly optimized, target-specific implementations of the library functions come at a high engineering cost: languages and compilers have failed to deliver performances competitive with expert written code, notably on GPUs and other hardware accelerators. Moreover, library implementations may not offer optimal performance for a specific use case. They may lack inter-operator optimizations and specializations to specific data sizes and shapes.

In his thesis, Ulysse Beaugnon, a former PhD student in the team, proposed to formulate this compilation problem as an optimization research problem using a combination of analytical modeling, experimental search, constraint programming and branch-and-bound optimization techniques. Basile Clement started a PhD to extend this idea, exploring the improvements required to make it fully competitive with handwritten code. In 2019, he evaluated reinforcement learning techniques such as multi-armed bandit schemes to improve the performance and efficiency of the search procedure; extended the analytical model with generic sizes, making it more precise before selecting tiling parameters; and made various improvements to the code generation procedure.

7. Bilateral Contracts and Grants with Industry

7.1. Bilateral Contracts with Industry

7.1.1. Collaboration with Airbus

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

7.2. Bilateral Grants with Industry

7.2.1. Google Research Fellowship: DWARF unwinding

Francesco Zappa Nardelli benefits from a Google Research Fellowship to pursue the work on DWARF unwinding, about 50k euros.

8. Partnerships and Cooperations

8.1. National Initiatives

8.1.1. ANR

The ANR JCJC project "FidelR" was awarded to Timothy Bourke this year and will begin in 2020.

8.1.1.1. ANR/CHIST-ERA DIVIDEND project, 2013-2019.

This project continues.

8.1.2. FUI: Fonds unique interministériel

8.1.2.1. 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).

8.1.3. Programme d'Investissements d'Avenir (PIA)

8.1.3.1. 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).

8.1.4. Others

8.1.4.1. 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.

8.2. European Initiatives

8.2.1. FP7 & H2020 Projects

- MNEMOSENE is a project with funding from the European Union's Horizon 2020 Research and Innovation Programme. Its objectives include the improvement of the energy-delay product, the computational efficiency and performance density by several orders of magnitude compared to state-of-the-art architectures. A cornerstone of the proposed solution is the memristor-based Compute-in-Memory (CIM) architecture, which eliminates long-distance, high-latency data transfers between memory and computing units required in conventional Von Neumann-based architectures by carrying out computations for performance-critical operations directly in memory.

- TETRAMAX, *Technology Transfer via Multinational Application Experiments*, 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.

8.3. International Initiatives

8.3.1. Participation in Other International Programs

- VerticA (Francesco Zappa Nardelli), 2017-2020, joint project with Northeastern University, USA, financed by the ONR (Office of Naval Research), $1.5M (subcontract for $150k).

9. Dissemination

9.1. Promoting Scientific Activities

9.1.1. Scientific Events: Selection

9.1.1.1. Chair of Conference Program Committees

- T. Bourke co-chaired the program committee of the ACM/IEEE international conference on Embedded Software (EMSOFT 2019).

9.1.1.2. Member of the Conference Program Committees

- T. Bourke served on the PC of the Euromicro Conference on Real-Time Systems (ECRTS 2019).
- T. Bourke served on the PC of the international workshop on Software and Compilers for Embedded Systems (SCOPES 2019).
- T. Bourke served on the PC of the international Modelica conference (MODELICA 2019).
- T. Bourke served on the PC of the OCaml Users and Developers Meeting (OCaml 2019).
- M. Pouzet served on the PC of the program committee of the ACM/IEEE international conference on Embedded Software (EMSOFT 2019).
- M. Pouzet served on the PC of the international workshop on Software and Compilers for Embedded Systems (SCOPES 2019).
- M. Pouzet served on the PC of the international workshop on Cyber-Physical Systems (CyPhy), a satellite event of ESWEEK.
- F. Zappa Nardelli served on the PC of the international conference on Object-Oriented Programming, Systems, Languages & Applications (OOPSLA 2019).
- F. Zappa Nardelli served on the PC of the ACM Workshop on Gradual Types (WGC 2020).

9.1.1.3. Reviewer

- T. Bourke reviewed submissions for the international conference on Computer Aided Verification (CAV 2019)
- T. Bourke reviewed submisssions for the international conference on Object-Oriented Programming, Systems, Languages & Applications (OOPSLA 2019).

9.1.2. Journal

9.1.2.1. Reviewer - Reviewing Activities

- T. Bourke reviewed articles for the Journal of Logical and Algebraic Methods in Programming (JLAMP).

9.1.3. Research Administration

- F. Zappa Nardelli chaired the part-time assistant professor recruitment committee at École Polytechinque.

9.2. Teaching - Supervision - Juries

9.2.1. 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.

Licence : T. Bourke, “Digital Systems” (L3), Lectures and TDs, ENS, France

Master: M. Pouzet & T. Bourke: “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, lab classes for “A Programmer’s introduction to Computer Architectures and Operating Systems" (M1), École Polytechnique, France

Master: F. Zappa Nardelli: “A Programmer’s introduction to Computer Architectures and Operating Systems" (M1), 45h, École Polytechnique, France

Bachelor: F. Zappa Nardelli: “A Programmer’s introduction to Computer Architectures and Operating Systems", 20h, École Polytechnique, France

Internships T. Bourke participated in reviewing the L3 and M1 internships of students at the ENS, France.

M1 Projects: T. Bourke is directing two M1 research projects.

9.2.2. Supervision

- PhD in progress: Lélio Brun, under review, supervised by T. Bourke and M. Pouzet.
- PhD in progress: Ismail Lakhim-Bennani, 1st year, supervised by M. Pouzet, G. Frehse, and T. Bourke
- PhD in progress: Basile Clément, 1st year, supervised by F. Zappa Nardelli and M. Pouzet.
- PhD: Chandan Reddy, supervised by A. Cohen, defended in March 2019.
- PhD: Ulysse Beaugnon, supervised by A. Cohen and M. Pouzet, defended in June 2019.

9.2.3. Juries

- T. Bourke was a jury member for the for the PhD thesis of Steven Varoumas, November 2019.
- F. Zappa Nardelli was a jury member for the PhD thesis of Francois Ginraud, Grenoble, Jan 2019.

9.3. Popularization

9.3.1. Internal or external Inria responsibilities

- T. Bourke participated in the *commission d’emplois scientifiques* (postdocs, delegations, and PhDs) for the Inria Paris research centre.
- T. Bourke participated in the PhD progress committees (*suivi doctorale*) for B. Medeiros de Barros and N. Kulatova.

- T. Bourke and M. Pouzet participated in meetings with the *Ministère de la Transition écologique et solidaire* on the validation of autonomous driving systems.

10. Bibliography

Major publications by the team in recent years

[1] T. Bourke, L. Brun, P.-E. Dagand, X. Leroy, M. Pouzet, L. Rieg. *A Formally Verified Compiler for Lustre*, in "PLDI 2017 - 38th ACM SIGPLAN Conference on Programming Language Design and Implementation", Barcelone, Spain, ACM, June 2017, https://hal.inria.fr/hal-01512286

[2] T. Bourke, F. Carcenac, J.-L. Colaço, B. Pagano, C. Pasteur, M. Pouzet. *A Synchronous Look at the Simulink Standard Library*, in "EMSOFT 2017 - 17th International Conference on Embedded Software", Seoul, South Korea, ACM Press, October 2017, 23 p. , https://hal.inria.fr/hal-01575631

[3] T. Bourke, J.-L. Colaço, B. Pagano, C. Pasteur, M. Pouzet. *A Synchronous-based Code Generator For Explicit Hybrid Systems Languages*, in "International Conference on Compiler Construction (CC)", London, United Kingdom, LNCS, July 2015, https://hal.inria.fr/hal-01242732

[4] L. Gérard, A. Guatto, C. Pasteur, 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], https://hal.inria.fr/hal-00728527

[5] J. C. Juega, S. Verdoolaege, A. Cohen, J. I. Gómez, C. Tenllado, F. Catthoor. *Patterns for parallel programming on GPUs*, in "Patterns for parallel programming on GPUs", F. Magoulès (editor), Saxe-Cobourg, 2013, vol. Evaluation of State-of-the-Art Parallelizing Compilers Generating CUDA Code for Heterogeneous CPU/GPU Computing, ISBN 978-1-874672-57-9, https://hal.archives-ouvertes.fr/hal-01257261

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

[7] R. Morisset, P. Pawan, 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], https://hal.inria.fr/hal-00909083

[8] A. Pop, A. Cohen. *OpenStream: Expressiveness and Data-Flow Compilation of OpenMP Streaming Programs*, in "ACM Transactions on Architecture and Code Optimization", 2013, vol. 9, n^{o} 4, Selected for presentation at the HiPEAC 2013 Conf [*DOI :* 10.1145/2400682.2400712], https://hal.inria.fr/hal-00786675

[9] J. Sevcik, V. Vafeiadis, F. Zappa Nardelli, S. Jagannathan, P. Sewell. *CompCertTSO: A Verified Compiler for Relaxed-Memory Concurrency*, in "Journal of the ACM (JACM)", 2013, vol. 60, n^{o} 3, pp. art. 22:1-50 [*DOI :* 10.1145/2487241.2487248], https://hal.inria.fr/hal-00909076

[10] V. VAFEIADIS, T. BALABONSKI, S. CHAKRABORTY, R. MORISSET, 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, January 2015, https://hal.inria.fr/hal-01089047

Publications of the year

Doctoral Dissertations and Habilitation Theses

[11] U. BEAUGNON. *Efficient Code Generation for Hardware Accelerators by Refining Partially Specified Implementations*, Ecole Normale Superieure de Paris - ENS Paris, June 2019, https://hal.archives-ouvertes.fr/tel-02385303

[12] C. REDDY. *Polyhedral Compilation for Domain Specific Languages*, Ecole normale supérieure, March 2019, https://hal.inria.fr/tel-02385670

Articles in International Peer-Reviewed Journals

[13] T. BASTIAN, S. KELL, F. ZAPPA NARDELLI. *Reliable and Fast DWARF-Based Stack Unwinding*, in "Proceedings of the ACM on Programming Languages", 2019, vol. OOPSLA [*DOI :* 10.1145/3360572], https://hal.inria.fr/hal-02297690

[14] T. BOURKE, L. BRUN, M. POUZET. *Mechanized semantics and verified compilation for a dataflow synchronous language with reset*, in "Proceedings of the ACM on Programming Languages", January 2020, vol. 4, n^{o} POPL, pp. 1-29 [*DOI :* 10.1145/3371112], https://hal.inria.fr/hal-02426573

[15] K. DIDIER, D. POTOP-BUTUCARU, G. IOOSS, A. COHEN, J. SOUYRIS, P. BAUFRETON, A. GRAILLAT. *Correct-by-Construction Parallelization of Hard Real-Time Avionics Applications on Off-the-Shelf Predictable Hardware*, in "ACM Transactions on Architecture and Code Optimization", August 2019, vol. 16, n^{o} 3, pp. 1-27 [*DOI :* 10.1145/3328799], https://hal.inria.fr/hal-02422789

Invited Conferences

[16] G. BAUDART, T. BOURKE, M. POUZET. *Symbolic Simulation of Dataflow Synchronous Programs with Timers*, in "12th Forum on Specification and Design Languages (FDL 2017)", Vérone, Italy, D. GROSSE, S. VINCO, H. PATEL (editors), Languages, Design Methods, and Tools for Electronic System Design: Selected Contributions from FDL 2017, Springer, January 2019, vol. 530, 25 p. [*DOI :* 10.1007/978-3-030-02215-0_3], https://hal.inria.fr/hal-01575621

International Conferences with Proceedings

[17] B. CHUNG, F. ZAPPA NARDELLI, J. VITEK. *Julia's efficient algorithm for subtyping unions and covariant tuples (Pearl)*, in "ECOOP 2019 - 33rd European Conference of Object-Oriented Programming", London, United Kingdom, July 2019 [*DOI :* 10.4230/LIPIcs.ECOOP.2019.6], https://hal.inria.fr/hal-02297696

[18] K. DIDIER, A. COHEN, D. POTOP-BUTUCARU, A. GAUFFRIAU. *Sheep in wolf's Clothing: Implementation Models for Dataflow Multi-Threaded Software*, in "ACSD 2019 - 19th International Conference on Application of Concurrency to System Design", Aachen, Germany, IEEE, June 2019, pp. 43-52 [*DOI :* 10.1109/ACSD.2019.00009], https://hal.inria.fr/hal-02422787

[19] J. PROY, K. HEYDEMANN, A. BERZATI, F. MAJERIC, A. COHEN. *A First ISA-Level Characterization of EM Pulse Effects on Superscalar Microarchitectures*, in "ARES 2019 - 14th International Conference on Availability, Reliability and Security", Canterbury, United Kingdom, ACM Press, 2019, pp. 7:1–7:10 [*DOI :* 10.1145/3339252.3339253], https://hal.sorbonne-universite.fr/hal-02373088

National Conferences with Proceedings

[20] T. BOURKE, M. POUZET. *Clocked arguments in a verified Lustre compiler*, in "JFLA 2019 - Les Trentièmes Journées Francophones des Langages Applicatifs", Les Rousses, France, Les actes des trentièmes Journées Francophones des Langages Applicatifs (JFLA 2019), January 2019, 16 p. , https://hal.inria.fr/hal-02005639

Conferences without Proceedings

[21] G. BAUDART, L. MANDEL, M. POUZET, E. ATKINSON, B. SHERMAN, M. CARBIN. *Programmation d'Applications Réactives Probabilistes*, in "JLFA 2020 - Journées Francophones des Langages Applicatifs", Gruissan, France, January 2020, https://hal.inria.fr/hal-02430070

[22] N. M. NOBRE, A. DREBES, G. RILEY, A. POP. *Bounded Stream Scheduling in Polyhedral OpenStream*, in "IMPACT 2020 - 10th International Workshop on Polyhedral Compilation Techniques", Bologna, Italy, January 2020, https://hal.inria.fr/hal-02441182

Scientific Books (or Scientific Book chapters)

[23] A. BENVENISTE, B. CAILLAUD, H. ELMQVIST, K. GHORBAL, M. OTTER, M. POUZET. *Multi-Mode DAE Models - Challenges, Theory and Implementation*, in "Computing and Software Science: State of the Art and Perspectives", Lecture Notes in Computer Science, October 2019, vol. 10000, pp. 283-310 [*DOI :* 10.1007/978-3-319-91908-9_16], https://hal.inria.fr/hal-02333603

Other Publications

[24] G. BAUDART, L. MANDEL, E. ATKINSON, B. SHERMAN, M. POUZET, M. CARBIN. *Reactive Probabilistic Programming*, August 2019, https://arxiv.org/abs/1908.07563 - working paper or preprint, https://hal.inria.fr/hal-02426533

[25] A. DREBES, L. CHELINI, O. ZINENKO, A. COHEN, H. CORPORAAL, T. GROSSER, K. VADIVEL, N. VASILACHE. *TC-CIM: Empowering Tensor Comprehensions for Computing-In-Memory*, January 2020, IMPACT 2020 - 10th International Workshop on Polyhedral Compilation Techniques, https://hal.inria.fr/hal-02441163

[26] N. M. NOBRE, A. DREBES, G. RILEY, A. POP. *Beyond Polyhedral Analysis of OpenStream Programs*, January 2019, 9th International Workshop on Polyhedral Compilation Techniques, https://hal.inria.fr/hal-02370558

[27] J. PROY, K. HEYDEMANN, F. MAJERIC, A. COHEN, A. BERZATI. *Studying EM Pulse Effects on Superscalar Microarchitectures at ISA Level*, April 2019, https://arxiv.org/abs/1903.02623 - working paper or preprint, https://hal.sorbonne-universite.fr/hal-02102373

References in notes

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

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

[30] P. Caspi, J.-L. Colaço, L. Gérard, M. Pouzet, 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

[31] A. Cohen, M. Duranton, C. Eisenbeis, C. Pagetti, F. Plateau, 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, January 2006

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

[33] J. Forget. *Un Langage Synchrone pour les Systèmes Embarqués Critiques Soumis à des Contraintes Temps Réel Multiples*, Université de Toulouse, November 2009

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

[35] L. Mandel, F. Plateau, 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, http://www.lri.fr/~mandel/papiers/MandelPlateauPouzet-MPC-10.pdf

[36] L. Mandel, F. Plateau, 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, October 30 – November 2 2011

[37] F. Plateau. *Modèle n-synchrone pour la programmation de réseaux de Kahn à mémoire bornée*, Université Paris-Sud 11, Orsay, France, 6 janvier 2010, https://www.lri.fr/~mandel/lucy-n/~plateau/these/

[38] M. Pouzet, 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, October 2009

[39] N. Vasilache, O. Zinenko, T. Theodoridis, P. Goyal, Z. Devito, W. S. Moses, S. Verdoolaege, A. Adams, 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)", October 2019, vol. 16, n^o 4, Article 38 p. [*DOI :* 10.1145/3355606]

[40] O. Zinenko, L. Chelini, T. Grosser. *Declarative Transformations in the Polyhedral Model*, Inria ; ENS Paris - Ecole Normale Supérieure de Paris ; ETH Zurich ; TU Delft ; IBM Zürich, December 2018, n^o RR-9243, https://hal.inria.fr/hal-01965599