Team PARKAS
Synchronous Kahn Parallelism


– 4 permanent researchers:
  
  Marc Pouzet    Albert Cohen    Jean Vuillemin    Louis Mandel

  Prof. P6/ENS, leader    DR, INRIA    Prof., ENS    MdC P11/ENS

– 8 PhD students, 1 post-doc

http://www.di.ens.fr/ParkasTeam.html
Scientific Objectives

How to program, in a mathematically-defined language:
- an embedded real-time controller?
- a model of its physical environment?
- a computing system running on a multi-core architecture?

- Program/test/verify/simulate/compile before the system is implemented,
  the source code serving for:
  - static analysis and simulation;
  - entry for the code generation of sequential and parallel code.

- Ensure strong properties of safety/efficiency at compile-time:
  - is the generated code equivalent to the source code?
  - does-it fulfill resource constraints (time, memory)?

A very modern trend in embedded system design: synchronous languages (e.g., Esterel, Lustre/SCADE, Signal), simulation languages (e.g., Simulink, LabView, Modelica).
SAO (Spécification Assistée par Ordinateur) — Airbus 80’s

Describe the system as block diagrams (synchronous communicating machines)
SCADE (Safety Critical Application Development Env. – Esterel-Tech.)

From computer assisted drawings to executable (sequential/parallel) code
Simulink/StateFlow – MathWorks

**Automata**

**Dataflow**

**Simulink Flowchart**

- `validate_sample_time` input
- `est_airflow` output
- `fb_correction` output
- `fuel_rate` output

**Feedback Control**

- Sensors
- Speed
- Throttle
- O2

**Feedforward Control**

- Pressure
- Throttle
- Speed
- Fail
- Multi

**Fueling Mode**

- Fueldisabled
- Low
- Rich
- Normal
- Overspeed
- Shutdown
- Running

**Oxygen Sensor Switching Threshold**

- 0.5

**Throttle Transient**

- \(0.01 - 0.01z^{-1}\)
- \(1 - 0.8z^{-1}\)

**Ramp Rate Ki**

- \(2 - D T_1 u_1 u_2\)
- \(2 - D T_2 u_1 u_2\)

**Pumping Constant**

- \(K_s T_1 z^{-1}\)
Software Factory Catia with LCM
(Delmia/Dassault-Systèmes)

- a wider domain than embedded system: simulate a whole factory
- a **Lucid Synchrone** compiler developed at DS integrated to Delmia automation
Programming massively parallel processors

- **data-flow computation**: language, hardware, parallel run-time.
- joint exploitation of data and pipeline parallelism, fine-grain (vector, simultaneous threads) or coarse grain (multi-core)

```
input/output (list)
list ::= list, item
    | item
item ::= stream
    | stream >> window
    | stream << window
stream ::= var
    | array[expr]
expr ::= var
    | value
```
Building compilers and program optimization

- **intermediate representation** in the polyhedral model, process networks and transformations of those described by sets of affine inequalities
- implementation in GCC, LLVM, XL at IBM, R-Stream at Reservoir Labs
Synchronous Data-flow Languages

The idea of Lustre (Caspi & Halbwachs, 1984):

- Write directly stream equations considered as executable specifications.
- Associate a compiler and static analysis tools to generate target embedded code.

E.g, a linear filter:

\[ f_0 = 0 \quad \forall n \in \mathbb{N}^*, f_n = f_{n-1} + s_n \quad s_n = 0.2(x_n - f_{n-1}) \]

is written (in Lucid Synchrone syntax):

```plaintext
let node filter(x) = f where
    rec f = 0.0 -> pre f +. s and s = 0.2 *. (x -. pre f)
```

- function composition; equations are time invariants
- The compiler ensures the absence of deadlock, determinism and execution in bounded time and space.
Background and Approach

In the PARKAS team, we work on:

- high-level programming languages for embedded systems: design, semantics, implementation
- compilation and parallel programming: internal representation of compilers (polyhedral model), optimization
- theoretical models and tools to enable the design automation of new application domains: real-time video processing, hybrid modeling, etc.

We base our research on synchronous Kahn data-flow used as:

- a programming model for dealing with time and parallelism
- internal representations in optimizing compilers
- code generation down to sequential and parallel code.

Theoretical results are validated into prototypes of languages and compilers with a close collaboration with industry.

**Impact**: Several of our research results had strong influence on production tools: SCADE 6 at Esterel-Tech., Catia with LCM at Dassault-Systèmes, GCC, compilers at IBM and Reservoir Labs.
Research questions

Increase expressiveness and safety of languages; put in relation logical parallelism and physical parallelism

Programming
- Features from functional languages to increase modularity.
- Integrate deterministic parallelism into general-purpose languages.

Extensions of the synchronous model
- Take communication through bounded buffers (e.g., TVHD)
- Model communication/computation time, jitter: E.g., sampled systems communicating through shared memory (avionic, engines)
- Can we account for discrete and continuous behaviors?

Compilation
- Generate modular sequential code.
- Parallel code (distributed, multi-core).
- Can we develop a fully traceable code generator with proved correctness and efficiency?
Programming language questions

Increase the expressiveness and safety of languages.

- Links with ML-like languages (e.g., Ocaml) : higher-order, type inference

- Dedicated type systems :
  - Clock calculus to ensure that the program can be executed synchronously [ICFP’96, EMSOFT’03]
  - Insure the absence of deadlocks (causality loops) [ESOP’01]; initialization issues [STTT’04].

- Mix of data-flow and hierarchical automata [EMSOFT’05, EMSOFT’06].

All have been experimented into a “laboratory” language called Lucid Synchrone. Close collaboration since 2000 with the SCADE team (JL-Colaco, B. Pagano) from Esterel-Tech.
SCADE 6 – Esterel-Technologies

Reactive Programming (ReactiveML)

Simulation of sensor networks (VERIMAG/FT, 2006-2008)

- The whole system is reactive but not real-time (possible dynamic creation).
- Global simulation: nodes, interaction between them and the environment, simulation (display, costs, etc.)

Example: Sim. of the energy consumption in a sensor network.

Extensions of Synchronous Languages

- Real-time streaming video: relaxing synchrony to \( n \)-synchrony
- Hybrid modeling: mixing discrete-time and continuous-time signals
Extension: from synchrony to n-synchrony


Question: how to program a larger class of systems without losing the safety of synchronous systems (e.g., determinism, static guaranties of bounded time and memory)?

– communication through bounded FIFOs
– express (and exploit) periodic behavior when possible;
– model jitter, execution time;
– give more freedom to the compiler/optimizer.

Related Work: Latency Insensitive Designs (Carloni, De Simone, etc.); Elastic Circuits (Cortadella et al.); SDF/CSDF (Ed. Lee); Cyclic scheduling (Chretienne, Baccelli, Munier, etc.) Network Calculus (Boudec, Baccelli, etc.); Real-Time Calculus (Thiele et al.)
Typical example : Picture-in-Picture

Incrustation of a Standard Definition (SD) image in a High Definition (HD) one

- **downscaler**: reduction of an HD image (1920×1080 pixels) to an SD image (720×480 pixels)
- **when**: removal of a part of an HD image
- **merge**: incrustation of an SD image in an HD image

**Questions:**

- the activation paces of the downscaler and the merge nodes?
- buffer size needed between the downscaler and the merge nodes?
- delay introduced by the picture in picture in the video processing chain?
Too restrictive for video applications

let node f x = t where
rec t = buffer y + buffer z
and y = x when (0011)
and z = x when (01)

- streams should be synchronous
- adding buffer (by hand) difficult and error-prone
- compute it automatically and generate synchronous code

relax the associated clocking rules
n-Synchronous Kahn Networks

- Pure synchrony: equality of clocks

\[
H \vdash e_1 : ck \quad H \vdash e_2 : ck
\]

\[
H \vdash \text{op}(e_1, e_2) : ck
\]

- Relaxed Synchrony: adaptability of clocks

\[
H \vdash e : ck \quad ck <: ck'
\]

\[
\text{(SUB)} \quad H \vdash \text{buffer } e : ck'
\]
Clocks as infinite binary words

$$O_{w_1}(i) = \text{cumulative function of 1 in } w_1$$
Adaptability Relation

buffer size \( \text{size}(w_1, w_2) = \max_{i \in \mathbb{N}}(\mathcal{O}_{w_1}(i) - \mathcal{O}_{w_2}(i)) \)

adaptability \( w_1 <: w_2 \stackrel{\text{def}}{=} \exists n \in \mathbb{N}, \forall i, 0 \leq \mathcal{O}_{w_1}(i) - \mathcal{O}_{w_2}(i) \leq n \)
let node f x = t where
rec t = buffer y + buffer z
and y = x when (0011)
and z = x when (01)

α_x \rightarrow α_y \text{ such that } \begin{cases} \alpha_x \text{ on } (0011) & \leq \alpha_t \text{ on } (1) \\ \alpha_x \text{ on } (01) & \leq \alpha_t \text{ on } (1) \end{cases}

Adaptability constraints are transformed into linear constraints.

\begin{align*}
I_{ay}(1) &= -\frac{v_{size_{ay}}}{2} + 1 \\
I_{ay}(2) &= 1 \\
I_{ax}(1) &= 1 \\
I_{ax}(2) &= 0 \\
I_{ax}(3) &= 1 \\
I_{ax}(4) &= 0 \\
\text{source} &= 1
\end{align*}
Abstraction of Clocks

\[ a_1 = \langle 0, \frac{4}{5} \rangle \left( \frac{3}{5} \right) \]

\[ O_{w_1} \]

\[ \Delta^1 : r \times i + b^1 \]

\[ \Delta^0 : r \times i + b^0 \]

\[ \text{concr}(b^0, b^1, r) = \left\{ w \mid \begin{array}{l}
w[i] = 1 \implies O_w(i) \leq \Delta^1(i) \\
w[i] = 0 \implies O_w(i) \geq \Delta^0(i) \end{array} \right\} \]
Abstraction

buffer size \( \text{size} \sim (a_1, a_2) = \lfloor b_1^1 - b_0^0 \rfloor \)

adaptability \( a_1 \lhd \sim a_2 \iff (r_1 = r_2) \land (b_1^2 - b_0^1 < 1) \)
Lucy-n [MPC’10] : a language similar to Lustre with buffers
- prototype for the programming within the n-synchronous model [POPL’06]
- type system to guaranty communication by bounded buffers
  - clock abstraction [APLAS’08]
  - correctness proofs of the abstraction in Coq : 8 800 SLOC
- automatic computation at compile time of the buffer sizes
New and Important Scientific Challenges

Code generation and parallelism

Driving challenge: the simultaneous satisfaction of the following objectives:

- Every refinement step preserves the functional semantics of a data-flow synchronous program: e.g., the type-based, modular repartition of a program
- Desynchronizing the program is always correct (as a Kahn network): e.g., \(n\)-synchronous composition to account for real-time constraints and overlap communications with computations
- Formal methods for functional and behavioral verification are available at each refinement and mapping step: e.g., support for translation validation, compilation tracing for certification
- Modular compilation is a strong priority at all refinement steps

Programming language research vs. optimizing compiler research

Languages for embedded system design emphasize control and verification at every refinement of a program by an expert designer; these are highly appreciated properties for the design of intermediate languages of any optimizing compiler
Extensions of Synchronous Languages

- Real-time streaming video: relaxing synchrony to $n$-synchrony
- Hybrid modeling: mixing discrete-time and continuous-time signals
Motivation and Context

- **Hybrid modelers** allow to program both a (discrete) controller and its physical (continuous) environment in the very same language.
- We focus on **explicit** modelers
- A lot of result on the formal verification of the sub-class of **hybrid automata** but relatively few on **programming language questions**.

What is the problem?

- Hybrid modelers (e.g., Simulink) widely used but they lack a formally defined semantics and code generation.

A new approach:

- **Extend** a synchronous language where dataflow equations are mixed with ODE.
- Make it **conservative**, i.e., nothing must change for the discrete subset (same typing, same code generation).
- Static typing to **divide** discrete from continuous signals
- **Recycle** existing synchronous compilers and numerical solvers to execute them.
Parallel composition : homogeneous case

Two equations with discrete time:

\[ f = 0.0 \rightarrow \text{pre } f + s \text{ and } s = 0.2 \times (x - \text{pre } f) \]

and the initial value problem:

\[ \text{der}(y') = -9.81 \text{ init } 0.0 \text{ and } \text{der}(y) = y' \text{ init } 10.0 \]

The first program can be written in any synchronous language, e.g. Lustre.

\[ \forall n \in \mathbb{N}^*, f_n = f_{n-1} + s_n \text{ and } f_0 = 0 \quad \forall n \in \mathbb{N}, s_n = 0.2 \times (x_n - f_{n-1}) \]

The second program can be written in any hybrid modeler, e.g. SIMULINK.

\[ \forall t \in \mathbb{R}^+, y'(t) = 0.0 + \int_0^t -9.81 \, dt \]

\[ \forall t \in \mathbb{R}^+, y(t) = 10.0 + \int_0^t y'(t) \, dt \]

Parallel composition is clear since equations share the same time scale.
Parallel composition: heterogeneous case

Two equations: a signal defined at discrete instants, the other continuously.

\[
\text{der}(\text{time}) = 1.0 \ \text{init} \ 0.0 \ \text{and} \ x = 0.0 \ \text{fby} \ x + \text{time}
\]

or:

\[
x = 0.0 \ \text{fby} \ x + 1.0 \ \text{and} \ \text{der}(y) = x \ \text{init} \ 0.0
\]

It would be tempting to define the first equation as:

\[
\forall n \in \mathbb{N}, x_n = x_{n-1} + \text{time}(n)
\]

And the second as:

\[
\forall n \in \mathbb{N}^*, x_n = x_{n-1} + 1.0 \ \text{and} \ x_0 = 1.0
\]

\[
\forall t \in \mathbb{R}_+, y(t) = 0.0 + \int_0^t x(t) \, dt
\]

i.e., \( x(t) \) as a piecewise constant function from \( \mathbb{R}_+ \) to \( \mathbb{R}_+ \) with

\[
\forall t \in \mathbb{R}_+, x(t) = x_{\lfloor t \rfloor}.
\]

In both cases, this would be a mistake. \( x \) is defined on a discrete, logical time; \( \text{time} \) on a continuous, absolute time.
Equations with reset

Two independent groups of equations.

\[ \text{der}(p) = 1.0 \ \text{init} \ 0.0 \ \text{reset} \ 0.0 \ \text{every} \ \text{up}(p - 1.0) \]

and

\[ x = 0.0 \ \text{fby} \ x + p \]

and

\[ \text{der}(\text{time}) = 1.0 \ \text{init} \ 0.0 \]

and

\[ z = \text{up}(\sin (\text{freq} \times \text{time})) \]

Properly translated in Simulink, changing \textit{freq} changes the output of \textit{x}!

If \textit{f} is running on a continuous time basis, what would be the meaning of :

\[ y = f(x) \ \text{every} \ \text{up}(z) \ \text{init} \ 0 \]

All these programs are \textit{wrongly typed} and should be statically rejected.
Discrete vs Continuous time signals

A signal is discrete if it is activated on a discrete clock.

A clock is termed discrete if it has been declared so or if it is the result of a zero-crossing or a sub-sampling of a discrete clock. Otherwise, it is termed continuous.

Ensure it statically with a type language and system for a synchronous language extended with ODEs.

\[ \sigma ::= \forall \beta_1, \ldots, \beta_n. t \xrightarrow{k} t \]

\[ t ::= t \times t \mid \beta \mid bt \]

\[ k ::= D \mid C \mid A \]

\[ bt ::= \text{float} \mid \text{int} \mid \text{bool} \mid \text{zero} \]

-- Typing rules follow the classical Milner type-system
Non-standard Semantics [CDC’10, JCSS (2011)]

realss \quad \textit{non-standard reals}

\[ \mathbb{R} \quad \text{+ infinitesimals (}\partial\text{)} \quad \mathbb{R}^* \]

\[ \cdots < t - 3\partial < t - 2\partial < t - \partial < t < t + \partial < t + 2\partial < t + 3\partial < \cdots \]

- Base clock both \textbf{dense} and \textbf{discrete}; BaseClock = \{n\partial \mid n \in \mathbb{N}^*\}
- \forall t. \ \cdot t \text{ is the previous instant, } t^\bullet \text{ is the next instant}

\[
\begin{align*}
\text{integr}^\#(T)(s)(s_0)(t) &= s'(t) \quad \text{where} \\
s'(t) &= s_0(t) \quad \text{if } t = \text{min}(T) \\
s'(t) &= s'(\cdot t) + \partial s(\cdot t) \\
\text{up}^\#(T)(s)(t^\bullet) &= \text{true} \quad \text{if } (s(\cdot t) \leq 0) \land (s(t) > 0) \text{ and } (t \in T) \\
\text{up}^\#(T)(s)(t^\bullet) &= \text{false} \quad \text{otherwise}
\end{align*}
\]

\[ \cdots \quad \cdots \quad \cdots \]
Working prototype implementation [LCTES’11]

- Uses Sundials CVODE (LLNL) numerical solver
- Conservative w.r.t. a Lustre-like language

\[
\text{let hybrid bouncing}(x_0,y_0,x'_0,y'_0) = (x,y) \text{ where}
\]
\[
\quad \text{der}(x) = x' \text{ init } x_0
\]
\[
\quad \text{der}(x') = 0.0 \text{ init } x'_0
\]
\[
\quad \text{der}(y) = y' \text{ init } y_0
\]
\[
\quad \text{der}(y') = -g \text{ init } y'_0 \text{ reset } -0.9 \times \text{ last } y' \text{ every up}(-y)
\]

Open questions:
- Static detection of some Zeno-behaviors: causality analysis.
- Multi-solvers: repartition.
- The same for DAEs (E.g., Modelica).
Conclusion: current/future research directions

Language/programming
- Semantics, static typing, compilation of hybrid modelers
- Relaxed synchrony (e.g., buffer-synchrony, quasi-synchrony)
- Large scade (parallel) simulation of discrete systems with reproducible behavior

Compilation for sequential and parallel code
- Obtain provably correct code generator for Lustre (VeLuS in Coq)
- Generate parallel code for multi-core with proved efficiency
- Make internal representation of optimizing compiler evolve from Static Single Assignment (SSA) towards data-flow equations with synchronous composition.
- Make compilation traceable (mandatory to meet avionic certification)
Highlights

Dissemination
- Wide-audience conferences: POPL, PLDI, DAC, IEEE CDC, Supercomputing
- Specialized venues: ESOP, ICFP, APLAS, EMSOFT, CASES, LCTES, MPC, CGO, CC, SAS, PACT, HiPEAC, ICS

Funding
- Coordinating the Synchronics INRIA Large Scale Action
- Partner of the HiPEAC European network of excellence
- 3 European research grants (1400 k€ over four years)
- French projects funded by ANR and Ministry of Industry

Close collaboration with production teams from industry
- Esterel-Technologies: SCADE
- Dassault-Systèmes: Catia with LCM, Modelica
- IBM, AMD, ARM, STMicroelectronics, Reservoir Labs, Kalray: parallel programming and compilation

Software
- Strong commitment on software dissemination
- Experimental research and transfer with academic prototypes and production compilers (SCADE, GCC)