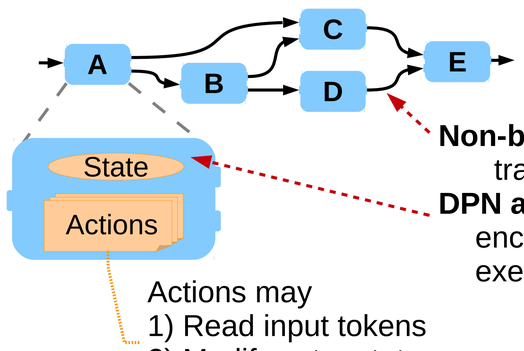
Efficient Software Synthesis of Dynamic Dataflow Programs

H. Yviquel[1], A. Sanchez[1], P. Jääskeläinen[2], J. Takala[2], M. Raulet[1], E. Casseau[3]
[1] INSA of Rennes, IETR, France
[2] Tampere University of Technology, Finland
[3] University of Rennes 1, IRISA, Inria, France

Dynamic Dataflow Modeling

Based on a Model of Computation introduced by Lee and Parks in 1995 and called Dataflow Process Network (DPN).



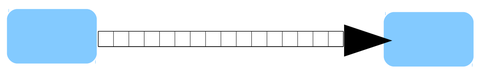
Non-blocking FIFO channels
transport discrete data (“tokens”)
DPN actors
encapsulate state (communicate only through ports)
execute actions sequentially according to firing rules

Actions may
1) Read input tokens
2) Modify actor state
3) Write output tokens

Efficient Software Synthesis

The synthesis translates dataflow descriptions into procedural code :

FIFO channels



Circular buffers

```
struct fifo_s {
  uint readIndex;
  uint writeIndex;
  type buffer[SIZE];
}
```

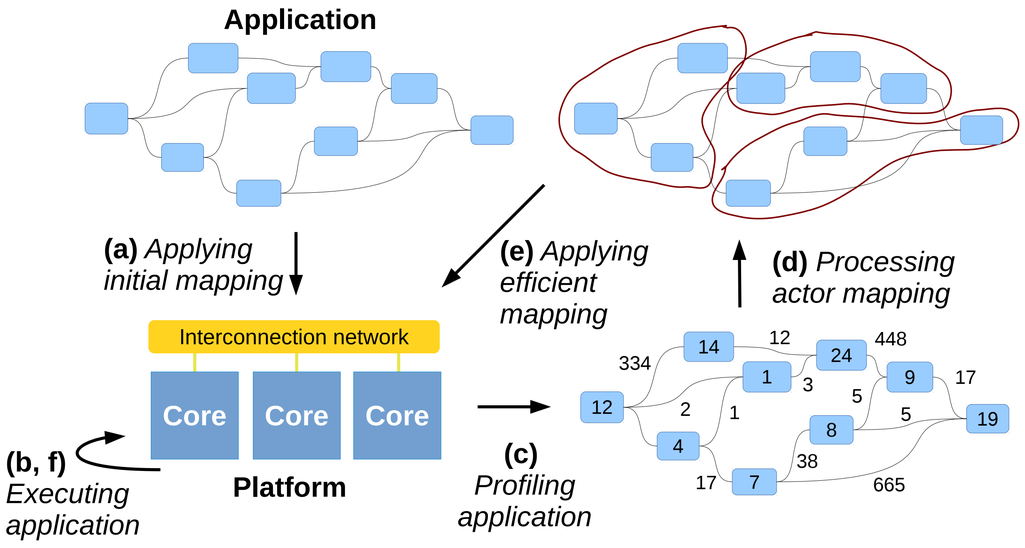
Actors
States
Firing rules (FSM, guards)
Actions (Read/write tokens)

Procedural tasks
Variables
Internal schedulers
Procedures (Access buffer + Update index)

The software synthesis of dynamic dataflow programs has been enhanced by :
- Using relative indexes for the circular buffer to **avoid costly conditional branching** when accessing to data.
- Computing directly from/to FIFO channel whenever it is possible: This removes additional data copies within muti-rate actions
- Detecting automatically when data are aligned in circular buffers: This detection **accelerates** FIFO accesses and allow **auto-vectorization** from the compiler.

Multicore Runtime

Our runtime allows the execution of **any dynamic dataflow programs** on multi-core platforms.



The runtime can be decomposed in 3 parts :
- A **lightweight profiling** is performed at runtime in order to determine the **computational loads** and the **communication rates** in the application.
- The **mapping system** assigns at runtime the actors to the processor cores according to the profiling results. As a result, the application can be **equitably balanced** on the platform. The mapping is only performed at predefined **synchronization point** to reduce the overhead of actor migration.
- The **distributed schedulers** order and time the actor execution on each processor core according to the **flow of data**. The actors are executed until they cannot fire anymore to benefit from **temporal** and **spatial locality**.

Orcc : Dataflow Programming Made Easy

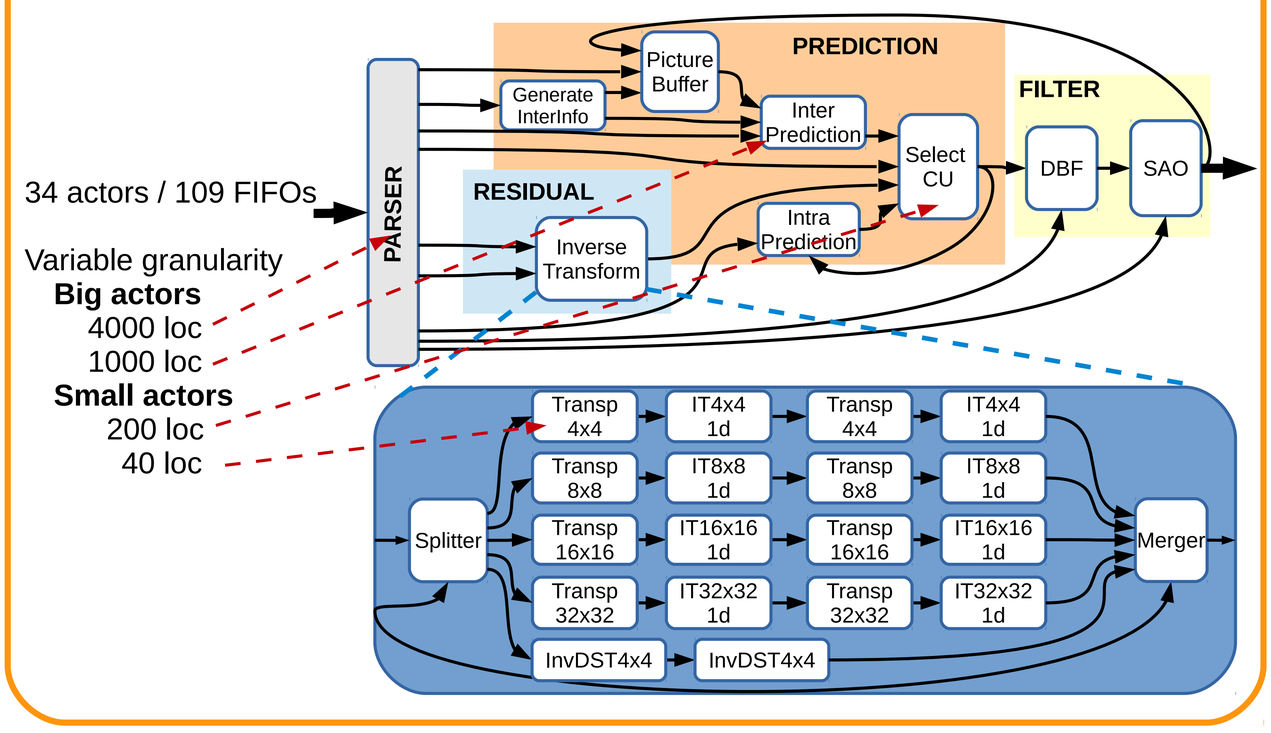
Orcc is an **open-source** Integrated Development Environment based on Eclipse and dedicated to dataflow programming.
- **Assisted writing of the applications:** A advanced editor for writing dataflow actors in CAL, and a intuitive graph editor for designing dataflow networks.
- **Fast debug and validation:** Orcc introduces innovative features for the debugging of dataflow programs, and integrates a simulator which allows quick functional verification.
- **Develop once, run everywhere:** The embedded compiler is able to generate both hardware or software code from a single description. Then, generated implementations can be executed on large panel of platforms (GPP, DSP, FPGA, etc) thanks to the available runtime libraries.

Freely available at http://orcc.sf.net

Reconfigurable Video Coding

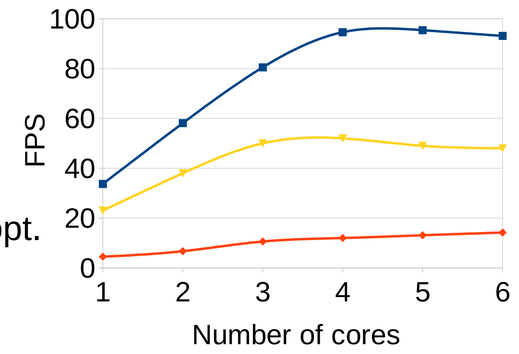
Reconfigurable Video Coding (RVC) is a **development framework** for video coding tools based on **dataflow programming**. The objectives of RVC are :
- Accelerating the standardization process of video coding technologies.
- Increasing flexibility of coding devices

The framework has been **standardized** by MPEG in 2009 and can be considered as the first **large-scale experimentation** on dynamic dataflow programming based on a subset of CAL Actor Language. Several **video decoders**, among other applications, have been developed using the RVC standard, such as the following HEVC decoder:



Results

Desktop implementation
Intel Xeon @ 3,2GHz // 720P sequence
MPEG-4 Visual: 95 FPS -- 3x (4 cores)
MPEG-4 AVC: 14 FPS -- 3x (6 cores)
HEVC: 52 FPS – 2x (4 cores)
Good frame-rate → High freq. / Assembly opt.
Limited speed-up → Communication cost



Embedded implementation
TTA multicore platforms with distributed mem.
Simulation @ 1GHz // 720P sequence
MPEG-4 Visual: 40 FPS -- 8x (16 cores)
HEVC: 5 FPS
Bad frame-rate → Low freq. / No opt.
Good speed-up → Pipeline parallelism