Synchronous Elastic Systems

Mike Kishinevsky and Jordi Cortadella

Intel
Strategic CAD Labs
Hillsboro, USA

Universitat Politecnica
de Catalunya
Barcelona, Spain
Contributors to SELF research

- Performance analysis: Jorge Júlvez
- Theory of elastic machines: Sava Krstic and John O’Leary
- Micro-architectural pipelining: Timothy Kam, Marc Galceran Oms
- Optimization: Dmitry Bufistov, Josep Carmona
- Bill Grundmann
Agenda

I. Basics of elastic systems

II. Early evaluation and performance analysis

III. Applications of elastic systems

IV. Demo of SELF compiler
Synchronous Stream of Data

Token (of data)

Clock cycle

... 7 4 1...

... 2 1 0...
Synchronous Elastic Stream

Clock cycle: 5 4 3 2 1 0

Bubble (no data)

Token
Synchronous Circuit

Latency = 0

\[ \begin{array}{c}
\cdots \cdots \begin{array}{c} 
7 \ 4 \ 1 \\
\end{array} + \\
\cdots \cdots \begin{array}{c} 
1 \ 0 \ 2 \\
\end{array} \rightarrow \begin{array}{c} 
8 \ 4 \ 3 \\
\end{array}
\end{array} \]
Synchronous Elastic Circuit

Latency = 0

Latency can vary
Ordinary Synchronous System

Changing latencies changes behavior
Synchronous Elastic
(characteristic property)

Changing latencies does NOT change behavior
= time elasticity
Elasticity?

- Elasticity refers to elasticity of time, i.e. tolerance to changes in timing parameters, not properties of materials.

- Luca Carloni et al. in the first systematic study of such systems called them Latency Insensitive Systems.

- Other used names:
  - Latency tolerant systems
  - Synchronous emulation of asynchronous systems
  - Synchronous handshake circuits

- We use term “synchronous elastic” to link to asynchronous elastic systems that have been developed before:
  - e.g., David Muller’s pipelines of late 1950s
  - Ivan Sutherland’s micro-pipelines 1989
  - Tolerate the variability of input data arrival and computation delays

- Asynchronous elastic tolerate changes in continuous time
  Synchronous elastic - in discrete time
Why

- Scalable
- Modular (Plug & Play)
- Better energy-delay trade-offs
  (design for typical case instead of worst case)
- New micro-architectural opportunities
  in digital design
- Not asynchronous: use existing design
  experience, CAD tools and flows... but have
  some advantages of asynchronous
How to Design Synchronous Elastic Systems

Example of the implementation:
SELF = Synchronous Elastic Flow

Others are possible
Pipelined communication

What if the sender does not always send valid data?
The Valid bit

What if the receiver is not always ready?
The Stop bit

sender

Data
Valid
Stop

receiver

Data
Valid
Stop

15
The Stop bit
The Stop bit

sender

Data

Valid

Stop

receiver

Data

Valid

Stop

17
The Stop bit

sender

Data

Valid

Stop

 receiver

Data

Valid

Stop

Back-pressure
The Stop bit

Long combinational path
Cyclic structures

One can build circuits with combinational cycles (constructive cycles by Berry), but synthesis and timing tools do not like them.
Example: pipelined linear communication chain with transparent latches

Master and slave latches with independent control
Shorthand notation
(clock lines not shown)
SELF (linear communication)
SELF

sender

Data

Valid

Stop

receiver

Data

Valid

Stop

1

0

En

V

S

En

V

S

En

V

S

En

V

S

Intel

28
SELF

sender

Data

Valid

Stop

receiver

Data

Valid

Stop
SELF

sender

Data

Valid

Stop

receiver

Data

Valid

Stop

1

1
SELF

sender

Data

Valid

Stop

receiver

Data

Valid

Stop

Data

Validity Signal

Stop Signal
SELF

sender

Data

Valid

Stop

receiver

Data

Valid

Stop

Intel

45
SELF

sender

Data

Valid

Stop

receiver

Data

Valid

Stop

En

En

En

En

V

V

V

V

1

0

S

S

S

S

Data Valid Stop

47
Elastic channel and its protocol

Sender

Data

Valid

Stop

Receiver

not Valid

Idle

Valid * not Stop

Transfer

Valid * Stop

Retry
Elastic channel protocol

Sender

<table>
<thead>
<tr>
<th>Data</th>
<th>Valid</th>
<th>Stop</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>D</td>
<td>D</td>
</tr>
<tr>
<td>*</td>
<td>D</td>
<td>C</td>
</tr>
<tr>
<td>C</td>
<td>C</td>
<td>C</td>
</tr>
<tr>
<td>C</td>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>B</td>
<td>A</td>
<td>A</td>
</tr>
</tbody>
</table>

Receiver

<table>
<thead>
<tr>
<th>Data</th>
<th>Valid</th>
<th>Stop</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Transfer

- Retry
- Idle

Intel

52
Basic VS block

VS block + data-path latch = elastic HALF-buffer (EHB)
EHB + EHB = elastic buffer with capacity 2
Control specification of the EB
Two implementations
Elastic buffer keeps data while stop is in flight

EBs = FIFOs with two parameters:
- Forward latency
- Capacity

Backward latency for stop propagation assumed (but need not be) equal to fwd latency

Typical case: (1,2) -
1 cycle forward latency with capacity of 2
- Replaces “normal” registers
- Decoupling buffers

W1R1 Cannot be done with
Single Edge Flops without double pumping

Can use latches inside Master-Slave as shown before
(Lazy) Fork
Eager Fork

Diagram showing logic gates with inputs V and S, and outputs S1, V1, V2, and S2.
Eager fork (another implementation)
Variable Latency Units

[Diagram showing the concept of variable latency units with a cloud labeled \([0 - k] \text{ cycles}\), arrows labeled 'go', 'done', and 'clear', and two 'V/S' blocks connected by a loop with a cycle symbol, all on a dark blue background.]
Coarse grain control
Elasticization

Synchronous  →  Elastic
Elastic control layer
Generation of gated clocks
Equivalence

**Synchronous:** stream of data

\[ D: \text{a b c d e d f g h i j} \ldots \]

**SELF:** elastic stream of data

\[
\begin{align*}
D: & \quad \text{a * b * * c d e * d f * g h * * i j} \ldots \\
V: & \quad 1 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 \\
S: & \quad 0 0 0 1 0 0 0 0 1 0 0 1 0 0 0 1 0 0 0 1 0 0 \\
\end{align*}
\]

Transfer sub-stream = original stream

Called: transfer equivalence, flow equivalence, or latency equivalence
Marked Graph models of elastic systems
Modelling elastic control with Petri nets
Modelling elastic control with Petri nets

Hiding internal transitions of elastic buffers
Modelling elastic control with Marked Graphs
Modelling elastic control with Marked Graphs

Forward
(Valid or Request)

Backward
(Stop or Acknowledgement)
Elastic control with Timed Marked Graphs.
Continuous time = asynchronous

\[ d = 250\text{ps} \quad \Rightarrow \quad d = 151\text{ps} \]

Delays in time units
Elastic control with Timed Marked Graphs. Discrete time = synchronous elastic
Elastic control with Timed Marked Graphs.
Discrete time. Multi-cycle operation

d=2  d=1

2  1
Elastic control with Timed Marked Graphs.
Discrete time. Variable latency operation

e.g. discrete probabilistic distribution:
average latency $0.8 \times 1 + 0.2 \times 2 = 1.2$
Modeling forks and joins

\[ d=1 \]
Modelling combinational elastic blocks
Elastic Marked Graphs

An Elastic Marked Graph (EMG) is a Timed MG such that for any arc $a$ there exists a complementary arc $a'$ satisfying the following condition:

- $a = a'$ and $a' = a$

Initial number of tokens on $a$ and $a'$ ($M_0(a) + M_0(a')$) = capacity of the corresponding elastic buffer

Similar forms of “pipelined” Petri Nets and Marked Graphs have been previously used for modeling pipelining in HW and SW (e.g. Patil 1974; Tsirlin, Rosenblum 1982)
Reminder: Performance analysis of Marked graphs

\[ Th = \text{operations / cycle} = \text{number of firings per time unit} \]

The throughput is given by the minimum mean-weight cycle

\[ Th = \min(Th(A), Th(B), Th(C)) = 2/5 \]

Efficient algorithms: (Karp 1978), (Dasdan, Gupta 1998)
Early evaluation

- Naïve solution: introduce choice places
  - issue tokens at choice node only into one (some) relevant path
  - problem: tokens can arrive to merge nodes out-of-order
    later token can overpass the earlier one

- Solution: change enabling rule
  - early evaluation
  - issue negative tokens to input places without tokens,
    i.e. keep the same firing rule
  - Add symmetric sub-channels with negative tokens
  - Negative tokens kill positive tokens when meet

- Two related problems:
  Early evaluation and Exceptions (how to kill a data-token)
Examples of early evaluation

MULTIPLEXOR

\[ \text{if } s = T \text{ then } c := a \quad \text{-- don’t wait for } b \]
\[ \text{else } c := b \quad \text{-- don’t wait for } a \]

MULTIPLIER

\[ \text{if } a = 0 \text{ then } c := 0 \quad \text{-- don’t wait for } b \]
Related work

Petri nets
- Extensions to model OR causality
  Kishinevsky et al. Change Diagrams [e.g. book of 1994]
  Yakovlev et al. Causal Nets 1996

Asynchronous systems
- Reese et al 2002: Early evaluation
- Brej 2003: Early evaluation with anti-tokens
- Ampalan & Singh 2006: preemption using anti-tokens
Dual Marked Graph

Marking: Arcs (places) $\rightarrow \mathbb{Z}$
(allow negative markings)

Some nodes are labeled as early-enabling

Enabling rules for a node:
- Positive enabling: $M(a) > 0$ for every input arc
- Early enabling (for early enabling nodes): $M(a) > 0$ for some input arcs
- Negative enabling: $M(a) < 0$ for every output arc

Firing rule: the same as in regular MG
Dual Marked Graphs

- Early enabling can be associated with an external guard that depends on data variables (e.g., a select signal of a multiplexor)
- Actual enabling guards are abstracted away (unless needed)
- **Anti-token generation**: When an early enabled node fires, it generates anti-tokens in the predecessor arcs that had no tokens
- **Anti-token propagation counterflow**: When negative enabled node fires, it propagates the anti-tokens from the successor to the predecessor arcs
Dual Marked Graph model
Passive anti-token

- Passive DMG = version of DMG without negative enabling
- Negative tokens can only be generated due to early enabling, but cannot propagate
- Let $D$ be a strongly connected DMG such that all cycles have positive cumulative marking
  Let $D_p$ be a corresponding passive DMG.

If environment (consumers) never generate negative tokens, then
throughput ($D$) = throughput ($D_p$)

- If capacity of input places for early enabling transitions is unlimited, then active anti-tokens do not improve performance
- Active anti-tokens reduce activity in the data-path (good for power reduction)
Properties of DMGs

- **Firing invariant:** Let node \( n \) be simultaneously positive (early) and negative enabled in marking \( M \). Let \( M_1 \) be the result of firing \( n \) from \( M \) due to positive (early) enabling. Let \( M_2 \) be the result of firing \( n \) from \( M \) due to negative enabling. Then, \( M_1 = M_2 \)

- **Token preservation.** Let \( c \) be a cycle of a strongly connected DMG with initial marking \( M_0 \). For every reachable marking \( M : M(c) = M_0(c) \)

- **Liveness.** A strongly connected passive DMG is live iff for every cycle \( c: M(c) > 0 \).
  - For DMGs this is a sufficient condition of liveness
  - It is also a necessary condition for positive liveness

- **Repetitive behavior.** In a SC DMG: a firing sequence \( s \) from \( M \) leads to the same marking iff every node fires in \( s \) the same number of times

- DMGs have properties similar to regular MGs
Implementing early enabling
How to implement anti-tokens?

Positive tokens

Negative tokens
How to implement anti-tokens?

Positive tokens

Negative tokens
How to implement anti-tokens?

Valid$^+$ → Valid$^+$
Stop$^+$ ← Stop$^+$
Valid$^-$ ← Valid$^-$
Stop$^-$ → Stop$^-$
Controller for elastic buffer
Dual controller for elastic buffer

En

\[ V^+ \]

\[ S^+ \]

\[ V^- \]

\[ S^- \]

\[ + \]

\[ \oplus \]

\[ \ominus \]
Dual Join and Fork
Join with early evaluation
Condition on Early Evaluation Function

Early evaluation function makes decision based on \textit{presence} of valid bits, not on their \textit{absence}.

Formally: EE is positive unate with respect to data input.

Example: legal EE function for a data-path MUX (s – select input)

\[
EE = V_{s}^{+} \land ((s \land V_{a}^{+}) \lor (\overline{s} \land V_{b}^{+}))
\]

\[
EE_{s} = V_{s}^{+} \land V_{a}^{+} \text{ and } EE_{\overline{s}} = V_{s}^{+} \land V_{b}^{+}
\]
Bigger capacity can be achieved by “injecting” anti-token up-down counters on elastic channels.
Properties of elastic channels

\[
\begin{align*}
\text{AG} \ ((V^+ \land S^+) \implies \text{AX} \ V^+) & \quad (\text{Retry}^+) \\
\text{AG} \ ((V^- \land S^-) \implies \text{AX} \ V^-) & \quad (\text{Retry}^-) \\
\text{AG} \ ((\overline{V^+} \lor \overline{S^-}) \land (\overline{V^-} \lor \overline{S^+})) & \quad (\text{Invariant (2)}) \\
\text{AG \ AF} \ ((V^+ \land \overline{S^+}) \lor (V^- \land \overline{S^-})) & \quad (\text{Liveness})
\end{align*}
\]

Invariants: mutually exclusive
- Kill \((V^-)\) and Stop \((S^+)\)
- Valid \((V^+)\) and retain of a kill \((S^-)\)
DLX processor model with slow bypass

Throughput: \( Th = \text{operations} / \text{cycle} \)

Late evaluation

\[ Th = 0.5 \]

Applying early evaluation on “Execution” and “Write-back”

\[ Th = 0.7 \quad (\alpha = 0.3; \beta = 0.3) \]
Conclusions

- Early evaluation can increase performance beyond the min cycle ratio.

- The duality between positive and negative tokens suggests a clean and effective implementation.

- Dual Marked Graphs is a formal model for analytical analysis and optimization methods.
Performance analysis with early evaluation

(joint work with Jorge Júlvez)
The throughput can also be computed by means of linear programming.

Average marking

\[ \overline{m}_p = \lim_{t \to \infty} \frac{1}{t} \int_0^t m_p(\tau) d\tau \]

Throughput

\[ th = \min_{p} \overline{m}_p \]

[Campos, Chiola, Silva 1991]
Revisit Performance Analysis of Marked Graphs

max \( th \)

\[
\begin{align*}
\overline{m}_{p1} &= 1 + t_b - t_a \\
\overline{m}_{p2} &= 0 + t_a - t_b \\
\overline{m}_{p3} &= 1 + t_d - t_a \\
\overline{m}_{p4} &= 0 + t_a - t_c \\
\overline{m}_{p5} &= 1 + t_c - t_d
\end{align*}
\]

reachability

th constraints

\[
\begin{align*}
\text{th} &\leq \overline{m}_{p2} \quad // \text{transition b} \\
\text{th} &\leq \overline{m}_{p4} \quad // \text{transition c} \\
\text{th} &\leq \overline{m}_{p5} \quad // \text{transition d} \\
\text{th} &\leq \min(\overline{m}_{p1}, \overline{m}_{p3}) \quad // \text{transition a}
\end{align*}
\]
GMG = Multi-guarded Dual Marked Graph

- Refinement of passive DMGs
- Every node has a set of guards
- Every guard is a set of input places (arcs)

Example:

```
G(t4) = \{(p1, p3), (p2, p3)\}
```
Early evaluation
Early evaluation

<table>
<thead>
<tr>
<th>(1 - \alpha)</th>
<th>(\alpha)</th>
<th>(\beta)</th>
<th>(1 - \beta)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0.43)</td>
<td>(0.60)</td>
<td>(0.40)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(\beta)</th>
<th>(\alpha)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>0.40</td>
</tr>
<tr>
<td>0.2</td>
<td>0.42</td>
</tr>
<tr>
<td>0.4</td>
<td>0.43</td>
</tr>
<tr>
<td>0.6</td>
<td>0.43</td>
</tr>
<tr>
<td>0.8</td>
<td>0.43</td>
</tr>
<tr>
<td>1.0</td>
<td>0.43</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(\beta)</th>
<th>(\alpha)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>0.40</td>
</tr>
<tr>
<td>0.2</td>
<td>0.42</td>
</tr>
<tr>
<td>0.4</td>
<td>0.43</td>
</tr>
<tr>
<td>0.6</td>
<td>0.43</td>
</tr>
<tr>
<td>0.8</td>
<td>0.43</td>
</tr>
<tr>
<td>1.0</td>
<td>0.43</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(\beta)</th>
<th>(\alpha)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>0.40</td>
</tr>
<tr>
<td>0.2</td>
<td>0.42</td>
</tr>
<tr>
<td>0.4</td>
<td>0.43</td>
</tr>
<tr>
<td>0.6</td>
<td>0.43</td>
</tr>
<tr>
<td>0.8</td>
<td>0.43</td>
</tr>
<tr>
<td>1.0</td>
<td>0.43</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(\beta)</th>
<th>(\alpha)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>0.40</td>
</tr>
<tr>
<td>0.2</td>
<td>0.42</td>
</tr>
<tr>
<td>0.4</td>
<td>0.43</td>
</tr>
<tr>
<td>0.6</td>
<td>0.43</td>
</tr>
<tr>
<td>0.8</td>
<td>0.43</td>
</tr>
<tr>
<td>1.0</td>
<td>0.43</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(\beta)</th>
<th>(\alpha)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>0.40</td>
</tr>
<tr>
<td>0.2</td>
<td>0.42</td>
</tr>
<tr>
<td>0.4</td>
<td>0.43</td>
</tr>
<tr>
<td>0.6</td>
<td>0.43</td>
</tr>
<tr>
<td>0.8</td>
<td>0.43</td>
</tr>
<tr>
<td>1.0</td>
<td>0.43</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(\beta)</th>
<th>(\alpha)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>0.40</td>
</tr>
<tr>
<td>0.2</td>
<td>0.42</td>
</tr>
<tr>
<td>0.4</td>
<td>0.43</td>
</tr>
<tr>
<td>0.6</td>
<td>0.43</td>
</tr>
<tr>
<td>0.8</td>
<td>0.43</td>
</tr>
<tr>
<td>1.0</td>
<td>0.43</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(\beta)</th>
<th>(\alpha)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>0.40</td>
</tr>
<tr>
<td>0.2</td>
<td>0.42</td>
</tr>
<tr>
<td>0.4</td>
<td>0.43</td>
</tr>
<tr>
<td>0.6</td>
<td>0.43</td>
</tr>
<tr>
<td>0.8</td>
<td>0.43</td>
</tr>
<tr>
<td>1.0</td>
<td>0.43</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(\beta)</th>
<th>(\alpha)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>0.40</td>
</tr>
<tr>
<td>0.2</td>
<td>0.42</td>
</tr>
<tr>
<td>0.4</td>
<td>0.43</td>
</tr>
<tr>
<td>0.6</td>
<td>0.43</td>
</tr>
<tr>
<td>0.8</td>
<td>0.43</td>
</tr>
<tr>
<td>1.0</td>
<td>0.43</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(\beta)</th>
<th>(\alpha)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>0.40</td>
</tr>
<tr>
<td>0.2</td>
<td>0.42</td>
</tr>
<tr>
<td>0.4</td>
<td>0.43</td>
</tr>
<tr>
<td>0.6</td>
<td>0.43</td>
</tr>
<tr>
<td>0.8</td>
<td>0.43</td>
</tr>
<tr>
<td>1.0</td>
<td>0.43</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(\beta)</th>
<th>(\alpha)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>0.40</td>
</tr>
<tr>
<td>0.2</td>
<td>0.42</td>
</tr>
<tr>
<td>0.4</td>
<td>0.43</td>
</tr>
<tr>
<td>0.6</td>
<td>0.43</td>
</tr>
<tr>
<td>0.8</td>
<td>0.43</td>
</tr>
<tr>
<td>1.0</td>
<td>0.43</td>
</tr>
</tbody>
</table>
LP formulation for an upper bound of a throughput (by example)

\[ \text{max} \ th \]

\[ \overline{m}_{p1} = 1 + t_b - t_a \]
\[ \overline{m}_{p2} = 0 + t_a - t_b \]
\[ \overline{m}_{p3} = 1 + t_d - t_a \]
\[ \overline{m}_{p4} = 0 + t_a - t_c \]
\[ \overline{m}_{p5} = 1 + t_c - t_d \]

\[ th \leq \overline{m}_{p2} \]
\[ th \leq \overline{m}_{p4} \]
\[ th \leq \overline{m}_{p5} \]

\[ th = \alpha \overline{m}_{p1} + (1-\alpha)\overline{m}_{p3} \]

\[ Th = \frac{(2 - \alpha)}{(3 - \alpha)} \]
Averaging cycle throughput or cycle times does not work

\[ T_h = \frac{2 - \alpha}{3 - \alpha} \]

Averaging throughput of individual cycles

\[ T_h' = \alpha \frac{1}{2} + (1 - \alpha) \frac{2}{3} = \frac{4 - \alpha}{6} \]

Averaging effective cycle times of individual cycles

\[ \frac{1}{T_h''} = 2\alpha + (1 - \alpha) \frac{3}{2} = \frac{3 + \alpha}{2} \]

\[ T_h'' = \frac{2}{3 + \alpha} \]

\[ T_h = \frac{2}{3 + \alpha} \]
What can we do with synchronous elastic systems?
Variable latency units

ALU

L = 1

ALU

L = 3
L = 2
L = 1

start

done
Benchmark "Patricia" from Media Bench

12 bits of an adder do 95% of additions

Statistics of operand sizes

12 bits of an adder do 95% of additions
Power-delay for an adder

Compare 64 bits VLA and prefix adder
Variable-latency cache hits

12-cycle miss

L2-cache

2-way associative
32KB
2-cycle hit

L1-cache

suggested by Joel Emer for ASIM experiment
Variable-latency cache hits

12-cycle miss

Pseudo-associative
32KB
{1-2} cycle hit

Sequential access: if hit in first access $L = 1$, if not – $L = 2$

Trade-off: faster, or larger, or less power cache
Variable-latency cache hits

12-cycle miss

Pseudo-associative
64KB
{2-3} cycle hit

Sequential access: if hit in first access L = 1, if not – L=2
Trade-off: faster, or larger, or less power cache
Correct-by-construction pipelining

Transforms:
- bypass
- retiming
- elasticize
- early enabling
- insert buffers and negative tokens
- size elastic buffer capacity

[Joint work with Timothy Kam and Marc Galceran]
Tree topology NoC

[In collaboration with Ken Stevens, Charles Dike, Bill Grundmann]
Router node interface

A

Router

B

C
Relative order of tokens between agents is preserved
Switch and Merge

\[
\text{Sel1} = \text{Vin1} \cdot (\text{P1} + \text{!Vin2}) \\
\text{Sel2} = \text{Vin2} \cdot (\text{!P1} + \text{!Vin1})
\]
Correctness (short story)

- Developed theory of elastic machines (for late evaluation)
- Verify correctness of any elastic implementation = check conformance with the definition of elastic machine
- All SELF controllers are verified for conformance
- Elasticization is correct-by-construction

- Theory for early evaluation and negative delays is more challenging
  - Sketch of a theory, but no fully satisfactory compositional properties found yet
  - Verification done on concrete systems and controllers
Summary

- SELF gives a low cost implementation of elastic machines
- Functionality is correct when latencies change
- New micro-architectural opportunities
- Compositional theory proving correctness
- Early evaluation - mechanism for performance and power optimization
- Optimization methods (that we did not discuss): Retiming and recycling, buffer optimization and pipelining
- Applications to design of NoC link layer
See reference list for some relevant publications
SELF compiler

Flow graph

Parameterized library of controllers

Control generation

Netlist of distributed controllers

Verilog

SMV

blif

Simulator

Performance

Backend synthesis

NuSMV

Verification

SIS & ABC

Logic synthesis
Example

![Diagram of a system with evaluation points and throughput values](image)

<table>
<thead>
<tr>
<th>Evaluation</th>
<th>Throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>No early evaluation</td>
<td>0.277</td>
</tr>
<tr>
<td>Passive anti-tokens M2 → W</td>
<td>0.280</td>
</tr>
<tr>
<td>Passive anti-tokens F3 → W</td>
<td>0.387</td>
</tr>
<tr>
<td>Active anti-tokens</td>
<td>0.400</td>
</tr>
</tbody>
</table>