Formal models for monotonic pipeline architectures

J.-P. Bodeveix, A. Bonenfant, T. Carle,
M. Filali, C. Rochange
IRIT Université de Toulouse France

TLA+ Community Meeting
May 4, 2025
Co-located with ETAPS 2025 in Hamilton, Ontario
Canada

Table of Contents

Plan

Real time systems

- Design, validation, certification.
- Hardware architectures.
- Accurate and *safe* Worst-Case Execution Time (WCET) bounds.

Issues

- Architectures to make tractable analyses.
- Validation of Architecture properties and assumptions.

Pipelined architectures

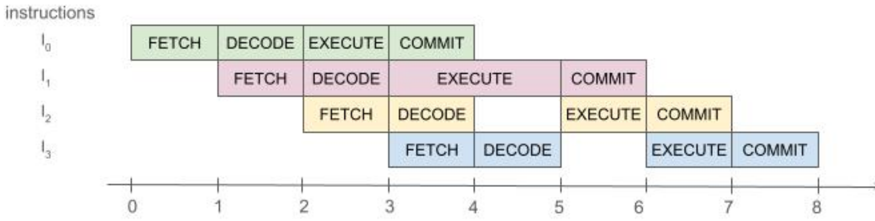

Figure: Execution of a 4-instruction sequence in a 4-stage pipeline. In this example, instruction I_2 depends on instruction I_1 which has a latency of 2 cycles in the EXECUTE stage. As a result, the execution of instruction I_2 is delayed by one cycle.

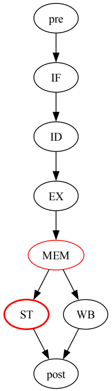

Figure: Hahn/Reineke architecture

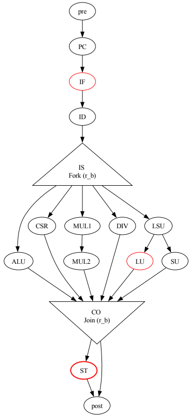

Figure: Minotaur architecture

Architectures timing anomalies

- A timing anomaly occurs when the usual assumptions to make analyses tractable are broken.
- example of a usual assumption:
  - a cache hit leads to a globally faster execution time.
  - a cache miss leads to a globally slower execution time.
- example of an anomaly:
  - a cache hit leads to a a globally *slower* execution time.
  - a cache miss leads to a globally *faster* execution time.

Basic anomalies:

- counter-intuitive anomaly.
- amplification anomaly.

PROTIPP project

Framework for the development of trustworthy generic hardware architectures

- basic components for absence of anomaly proofs.
- Use of the “right” tool for an easy development based of formal methods:
  - development: Event-B,
  - proofs: Coq, Isabelle HOL, TLA$^+$, Event-B,
  - model checking: TLA$^+$,
  - simulation (small): Coq.

Plan

Instructions and Stages

- Instructions are finite and totally ordered.
- Stages are finite, partially ordered and well-founded.
- The pipeline is modelled as an acyclic stage graph.
- To each instruction is assigned a path over this graph. *pre* is the first (virtual) stage, *post* is the last(virtual) stage.
- Delays:
  - Each stage is characterized by a given computation delay.
  - When visiting a stage, an instruction can access memory. This access can be a hit or a miss with given delays.

```
-------- MODULE Archi_stat -----------
EXTENDS FiniteSets, Naturals, Misc, Sequences, SequencesExt
        , WellFoundedInduction
CONSTANTS Stage, Inst, path, graph_TC, pre, post, mem, lat_st, lat_h,
     lat_m
ASSUME f_Stage ≜ IsFiniteSet(Stage)
ASSUME f_Inst ≜ IsFiniteSet(Inst)
ASSUME path_ty ≜ path ∈ [Inst → Seq(Stage)]
ASSUME graph_TC_ty ≜ graph_TC ∈ SUBSET (Stage × Stage)
ASSUME graph_TC_Trans ≜ IsTransitivelyClosedOn(graph_TC, Stage)
(* ASSUME graph_TC_Order ≜ IsWellFoundedOn(graph_TC, Stage) *)
ASSUME graph_path_min ≜
    ∀ e1,e2 ,b ∈ Stage,i ∈ Inst : ⟨e1,e2⟩ ∈ graph_inst(path,i)
       ⇒ (e2 # post ⇒ (⟨b,e2⟩ ∈ graph_TC ⇒ ⟨b,e1⟩ ∈ graph_TC ∨ b = e1
     ))
ASSUME graph_archi ≜ graph(path,Inst) ⊆ graph_TC
```

Generic sub-architectures

- at most one instruction by stage, no Join : (Hahn/Reineke).
- queues between stages, one fork-join (Minotaur)

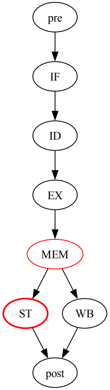

Figure: Hahn/Reineke architecture

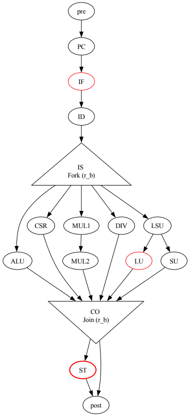

Figure: Minotaur architecture

Instructions move over the pipeline

- Instructions move between stages in a synchronous way.
- The global move is done according to node capacity and their readiness (e.g. termination of the current instruction).
- The move is maximal.

Remark: synchronous concurrency: deterministic.

```
------------- MODULE Archi_dyn ------------
EXTENDS Misc, Archi_stat, SequencesExt
CONSTANTS ready, hit
(*
ASSUME ready_ty ≜ ready ∈ [(TimedState × Inst) → BOOLEAN]
ASSUME ready_pre ≜
    ∀ St ∈ TimedState: ∀ r ∈ Inst :
        wd_St(St) ⇒ ({i ∈ Inst : stage(St,i) = pre} # ∅
          ⇒ (ready[St,r] ⇒ r = Min({i ∈ Inst : stage(St,i) = pre})))
ASSUME ready_post ≜ ∀ St ∈ TimedState, i ∈ Inst:
            wd_St(St) ⇒ (stage(St,i) = post ⇒ ¬ (ready[St,i]) )
ASSUME ready_cnt ≜ ∀ St ∈ TimedState, i ∈ Inst:
          wd_St(St) ⇒ (ready[St,i] ⇒ cnt(St,i) = 0)
```

The basic transition

```
next_cnt(St,i) ≜ IF cnt(St,i) > 0 THEN cnt(St,i) – 1 ELSE cnt(St,i)

Cycle[St ∈ TimedState] ≜ (* synchronous move *)
  [i ∈ Inst ↦
    IF stage(St,i) = post THEN St[i]
    ELSE IF ready[St,i] ∧ willbefree[St, next_stage(stage(St,i), i)] THEN
        ⟨ix(St,i) + 1, lat_p(next_stage(stage(St,i), i), i)⟩
    ELSE ⟨ix(St,i), next_cnt(St,i)⟩ (* compute or stall *)
  ]
```

(Hahn/Reineke).

```
willbefree [St ∈ TimedState, st ∈ Stage] ≜
    ∧ wd_St(St)
    ∧ ∨ st = post
      ∨ ¬ (∃ i ∈ Inst : stage(St,i) = st)
      ∨ ∃ i ∈ Inst :
            ∧ stage(St,i) = st
            ∧ ready[St,i]
            ∧ willbefree [St, next_stage(st,i)]
```

(Hahn/Reineke).

Remark. The definition relies on the wellfoundedness of stages.

The transition system

```
---------------- MODULE ts --------------------
EXTENDS Naturals, Sequences, Archi_dyn

VARIABLES clock, state

vars ≜ ⟨ clock,  state ⟩

Init  ≜ clock = 0 ∧ state = Pre (* [i ∈ Inst ↦ ⟨1,0⟩] *)

Next ≜
     ∧ state'  = Cycle[state]
     ∧ clock'  = IF state' = state THEN clock
                 ELSE clock + 1
```

Remark: The clock is for trace readability.

Plan

State ordering

```
(* progress order over instructions  stages and counters *)
St1_C1 ⊑ St2_C2 ≜
     ⟨St1_C1[1],St2_C2[1]⟩ ∈ graph_TC
  ∨ ( St1_C1[1]= St2_C2[1] ∧ St1_C1[2] ≥ St2_C2[2])
St1_C1 ⊏ St2_C2 ≜ St1_C1 ⊑ St2_C2 ∧ ¬ St2_C2 ⊑ St1_C1

(* Pipeline state progress *)
St1 ⪯ St2 ≜
   ∀ i ∈ Inst: ⟨stage(St1,i), cnt(St1,i)⟩ ⊑ ⟨stage(St2,i), cnt(St2,i)⟩
St1 ≺ St2 ≜ St1 ⪯ St2 ∧ ¬ St2 ⪯ St1
```

(Hahn/Reineke).

Positive progress (1)

Functional definitions

$$
\begin{array}{l}
\text{inorder\_exp}(i, j, st\_i, st\_j) \triangleq st\_j\#post \Rightarrow (\langle st\_i, st\_j\rangle \in graph\_TC \Rightarrow j < i) \\
\text{inorder}(St) \triangleq \forall\, i, j \in Inst:\ ix(St, i) \in steps(i) \wedge ix(St, j) \in steps(j) \\
\qquad\qquad \Rightarrow \text{inorder\_exp}(i, j, stage(St, i), stage(St, j)) \\
inP(St) \triangleq \{i \in Inst:\ stage(St, i)\ \#\ post\} \\
farthest(St) \triangleq Min(inP(St)) \\
Inv\_Min(St) \triangleq \\
\quad \vee\ inP(St) = \emptyset \\
\quad \vee\ \textbf{LET}\ n\_st\_m \triangleq next\_stage(stage(St, farthest(St)), farthest(St))\ \textbf{IN} \\
\qquad\quad n\_st\_m = post \vee \neg(\exists\, i \in Inst:\ stage(St, i) = n\_st\_m)
\end{array}
$$

Positive progress (2)

Dynamic properties

LEMMA next_stage_farthest $\triangleq \forall$ St $\in$ TimedState:
$\wedge$ wd_St(St) $\wedge$ inorder(St)
$\wedge$ inP(St) # $\emptyset$
$\wedge$ next_stage(stage(St,farthest(St)), farthest (St)) # post
$\Rightarrow \neg$ ($\exists$ i $\in$ Inst : stage(St,i)=next_stage(stage(St,farthest(St)), farthest (St)))

LEMMA STABLE_Inv_Min $\triangleq \forall$ St $\in$ TimedState:
wd_St(St) $\wedge$ inorder(St) $\wedge$ Inv_Min(St) $\Rightarrow$ Inv_Min(Cycle[St])

THEOREM PositiveProgress $\triangleq \forall$ St $\in$ TimedState:
$\wedge$ wd_St(St) $\wedge$ inorder(St) $\wedge$ Inv_Min(St)
$\wedge$ inP(St) # $\emptyset$
$\Rightarrow$ St $\prec$ Cycle[St]
=====================================

Monotonicity (1)

```
CONSTANTS hit1, hit2
ASSUME hit1_ty ≜ hit1 ∈ [Inst → Seq(BOOLEAN)]
ASSUME path_hit1 ≜ ∀ i ∈ Inst: Len(hit1[i]) = Len(path[i])
ASSUME hit2_ty ≜ hit2 ∈ [Inst → Seq(BOOLEAN)]
ASSUME path_hit2 ≜ ∀ i ∈ Inst: Len(hit2[i]) = Len(path[i])
ASSUME hit2_hit1 ≜
    ∀ i ∈ Inst : ∀ k ∈ DOMAIN (path[i]): hit2[ i ][ k] ⇒ hit1 [ i ][ k]
```

Monotonicity (2)

```
Inst1 ≜ INSTANCE ts WITH
          clock ← clock1,
          hit   ← hit1,
          state ← state1

Inst2 ≜ INSTANCE ts WITH
          clock ← clock2,
          hit   ← hit2,
          state ← state2

THEOREM monotonicity ≜
    ∧ HR_Invariant(state1)
    ∧ HR_Invariant(state2)
    ∧ state2 ≼ state1 ⇒ Cycle[state2] ≼ Cycle[state1]
```

Comments

The monotonicity is based on a generic invariant:

- Mutual exclusion inside a stage.
- Inorder property.
- general assumption about the ready predicate of specific invariant for HR architecture.

(under study)

Plan

Hahn-Reineke architecture

- Features:
  - At most one instruction on a stage.
  - No join node.
- First development in Event-B.
- Mechanization in Isabelle-HOL and TLA$^+$.

Plan

Conclusion (1)

- A generic architecture.
- Mechanization of the Hahn/Reineke case study.
  - Expression and proof of most of the properties over a generic architecture.
  - The case study is an instance of the generic architecture.
- Future work: Minotaur case study.
  - Stages have fifos.
  - Join node (scoreboard).
  - Architecture optimizations.

Conclusion (2)

- Formalization of a generic hardware pipeline environment for *experimentation*.
- Formalization of some hardware concepts and their properties over the experimentation environment.
  - Formalization of the necessary (or some sufficient) assumptions to prove these properties
  - Validation through model checking or simulation.

generic *invariants*