Mothers of Pipelines

Sava Krstić, Robert Jones, John O’ Leary
Intel Corporation

PDPAR 2006
Pipeline Verification: \( P \) vs. \( ISA \)

Prove that a high-level model \( P \) of a pipelined processor is faithful to the given instruction set architecture (\( ISA \)).

* Both \( P \) and \( ISA \) are seen as transition systems; the goal is to prove that \( ISA \) in some reasonable sense simulates \( P \).
**ISA**

**States:** \( \mathcal{I} = \langle pc : \text{lAddr}, rf : \text{RF}, dmem : \text{DMem}, imem : \text{IMem} \rangle \)

**Transitions:**

<table>
<thead>
<tr>
<th>guard: ( imem.pc = \ldots )</th>
<th>action</th>
</tr>
</thead>
<tbody>
<tr>
<td>( opc1 ) dest src1 src2</td>
<td>( pc := pc + 4 ) ( rf.\text{dest} := \text{alu} \ opc1 \ (rf.\text{src1}) \ (rf.\text{src2}) )</td>
</tr>
<tr>
<td>( opc2 ) dest src1 imm</td>
<td>( pc := pc + 4 ) ( rf.\text{dest} := \text{alu} \ opc2 \ (rf.\text{src1}) \ \text{imm} )</td>
</tr>
<tr>
<td>( \text{ld} ) dest src1 offset</td>
<td>( pc := pc + 4 ) ( rf.\text{dest} := \text{dmem} \ (rf.\text{src1} + \text{offset}) )</td>
</tr>
<tr>
<td>( \text{st} ) src1 dest offset</td>
<td>( pc := pc + 4 ) ( \text{dmem} \ (rf.\text{dest} + \text{offset}) := rf.\text{src1} )</td>
</tr>
<tr>
<td>( opc3 ) reg offset</td>
<td>( pc := \begin{cases} \text{br_target} &amp; \text{if } \text{br_taken} \ pc + 4 &amp; \text{otherwise} \end{cases} ), where ( \text{br_target} = \text{target} \ pc \ \text{offset} ) ( \text{br_taken} = \text{taken} \ \opc3 \ (rf.\text{reg}) )</td>
</tr>
</tbody>
</table>

\( \text{dest}, \text{src1}, \text{src2}, \text{reg} : \text{Reg} \quad \text{imm}, \text{offset} : \text{Word} \)

\( \opc1 : \{\text{add, sub, mult}\} \quad \opc2 : \{\text{addi, subi, multi}\} \quad \opc3 : \{\text{beqz, bnez, j}\} \)
Example: $P = DLX$

States of $P$: $\langle pc, rf, dmem, imem, p_1, p_2, p_3, p_4 \rangle$

[regular cycle]

1. $p_4$ writes back to rf and retires
2. $p_3$ does memory access
3. alu computes result/mem. address for $p_2$
4. $p_1$ gets data from rf or by forwarding;
in the branch case, target/taken computed
5. new $p'_1$ is fetched; pc incremented
6. like 4, plus updating pc with computed target

[branch taken]

[stall for load]
Simulating $P$ in $ISA$

First need to map states of $P$ to $ISA$ states:

$P$-state $= \{ p_1, p_2, p_3, p_4, pc, rf, dmem, imem \}$

$ISA$-state $= \{ pc^*, rf^*, dmem^*, imem^* \}$
Flushing

Mapping states of $P$ to ISA states:

$P$-state $= p_1, p_2, p_3, p_4$

ISA-state $= pc, rf, dmem, imem$

$\alpha$

$pc, rf, dmem, imem$

$pc', rf', dmem', imem$

$pc', rf', dmem', imem$

$pc', rf', dmem', imem$

$pc', rf', dmem', imem$

$pc', rf', dmem', imem$
The Burch-Dill Method for \textit{DLX} \hfill \textsuperscript{[CAV 94]}

- \( \mathcal{D} \) \hspace{1cm} \textit{DLX} states
- \( \text{dlx\_step} : \mathcal{D} \to \mathcal{D} \) \hspace{1cm} \textit{DLX} transitions
- \( \alpha : \mathcal{D} \to \mathcal{I} \) \hspace{1cm} flushing function

**Theorem**

\[ \begin{array}{ccc}
\mathcal{D} & \xrightarrow{\text{dlx\_step}} & \mathcal{D} \\
\downarrow \alpha & & \downarrow \alpha \\
\mathcal{I} & \xrightarrow{\text{isa\_step}} & \mathcal{I}
\end{array} \]
The Burch-Dill Method in General

- $\mathcal{P} = \mathcal{I} \times \text{PipeRegs}$ (states of $P$)
- $p_{\text{step}} : \mathcal{P} \rightarrow \mathcal{P}$ (transitions of $P$)
- $\alpha : \mathcal{P} \rightarrow \mathcal{I}$ (flushing function)

Correctness Theorem for $P$:

For proof, don’t need defs of alu, target, taken, and only need basic read-write array properties of $rf$ and $\text{mem}$

Correctness thm expressed in the language of SMT solvers

... but it’s huge for complex $P$. Must partition the thm.
The *MOP* Method

Put a highly-nondeterministic “machine” *MOP* between *P* and *ISA* and deduce the *P* vs. *ISA* correctness from *P* vs. *MOP* and *MOP* vs. *ISA*.

- *MOP* is derived from *ISA* and the features of *P*
- *MOP* is the mother of *many* pipelines:

\[ P_1 \ \rightarrow \ \cdots \ \rightarrow \ \leftrightarrow \ \cdots \ \rightarrow \ P_{n} \]

- Out-of-order execution and non-determinism in *P* are OK
- Potential to systematically partition correctness theorem
MOP: Starting Observations

- One can formalize “instructions-in-flight”—parcels
- Parcels form a poset of finite height—progress ordering
- With every state of $P$ we can associate a set of parcels
- At each cycle of execution of $P$:
  - $pc$, $rf$, $mem$ get updated
  - some parcels get in (fetched)
  - some parcels get out (retired)
  - some parcels progress
Parcels

\[
\text{Parcel} \ = \ \langle \\
\quad \text{instr} : \text{Instr}_\perp \\
\quad \text{my}_{-}\text{pc} : \text{IAddr}_\perp \\
\quad \text{dest, src1, src2} : \text{Reg}_\perp \\
\quad \text{imm} : \text{Word}_\perp \\
\quad \text{opc} : \text{Opcode}_\perp \\
\quad \text{data1, data2} : \text{Word}_\perp \\
\quad \text{res, mem\textunderscore addr} : \text{Word}_\perp \\
\quad \text{tkn} : \text{bool}_\perp \\
\quad \text{next\textunderscore pc} : \text{IAddr}_\perp \\
\quad \text{wb} : \{\perp, \top\} \\
\quad \text{pc\textunderscore upd} : \{\perp, \_, m, \top\}
\rangle
\]
Definition of \textit{MOP}

**States:** \( \mathcal{M} = \mathcal{I} \times \langle q : \mathbb{N} \rightarrow \text{Parcel}, \ head : \mathbb{N}, \ tail : \mathbb{N} \rangle \)

**Transitions:** Atomic actions occurring in executions
Transitions

\[
\text{def } \quad i = \text{imem.pc} \\
\text{grd } \quad \text{length} = 0 \lor q.\text{tail.pc_upd} \neq \bot \\
\text{act } \quad q.(\text{tail + 1}) := \text{emptyParcel}[\text{instr} \leftrightarrow i, \text{my_pc} \leftrightarrow \text{pc}] \quad \text{tail} := \text{tail + 1}
\]

\[
\text{def } \quad p = q.j \\
\text{grd } \quad \text{head} \leq j \leq \text{tail} \quad \neg(\text{decoded } p) \\
\text{act } \quad p := \text{decode } p
\]

* 16 more rules: data1_rf, data1_forward, result, mem_addr, write_back j, load j, store j, branch_target j, branch_taken j, next_pc_branch j, next_pc_nonbranch j, pc_update, speculate, prediction_ok j, squash, retire
Confluence

\(MOP^\# \equiv MOP \) without \texttt{fetch}\n
**Theorem** \(MOP^\#\) is terminating.

**Theorem** Both \(MOP\) and \(MOP^\#\) are locally confluent.

(\(\approx 400\) little theorems)
Local Confluence: About Proof

Most cases are resolved trivially:

Some are interesting:
Flushed and Burch-Dill for MOP

Define

- **flushed MOP state** \(\equiv\) its \(q\)-component is empty

Given \(m \in \mathcal{M}\), any MOP# run \(m \rightarrow m_1 \rightarrow m_2 \rightarrow \ldots\)

- \(\ldots\) terminates \(\ldots\)
- \(\ldots\) in the same final *flushed* state \(m'\)

Define

- \(\alpha(m) = \) the ISA-component of \(m'\)

Burch-Dill **Theorem** for MOP:

\[ \mathcal{M} \xrightarrow{\rho} \mathcal{M} \xrightarrow{\alpha} \mathcal{I} \xrightarrow{\alpha} \mathcal{I} \]

(for any MOP rule \(\rho\))
Simulating Microarchitectures in *MOP*

**Theorem** To verify \( P \) against ISA, it suffices to find \( \beta: \mathcal{P} \rightarrow \mathcal{M} \) such that, for every \( p \in P \), \( \beta \left( \text{p\_step } p \right) \) is reachable from \( \beta \ p \).

**Proof:**

![Diagram showing the proof of the theorem](image-url)
How to find a sequence of $MOP$ transitions from $m$ to $m'$?

—Using sequence of “quasi $P$-states” from $p$ to $p'$:

$$\langle v_1, v_2, v_3, \ldots, v_n \rangle = p$$

$$\rightsquigarrow \langle v'_1, v_2, v_3, \ldots, v_n \rangle = p_1$$

$$\rightsquigarrow \langle v'_1, v'_2, v_3, \ldots, v_n \rangle = p_2$$

$$\rightsquigarrow \ldots$$

$$\rightsquigarrow \langle v'_1, v'_2, v'_3, \ldots, v'_n \rangle = p_{n-1}$$

$$\rightsquigarrow \langle v'_1, v'_2, v'_3, \ldots, v'_n \rangle = p'$$

... get the corresponding $MOP$ states $m, m_1, \ldots, m_{n-1}, m'$ and short $MOP$ paths from $m_i$ to $m_{i+1}$.
What We’ve Done

- Models of \(ISA, MOP, DLX\) in \(reF\text{L}ect\)
- Local confluence proofs with CVCL
- Simulation proofs for \(DLX\) (via short paths in \(MOP\)) with CVCL
To Do

- Case study of an out-of-order processor model
- Refining the method
  - Systematic ways of defining $\beta : \mathcal{P} \rightarrow \mathcal{M}$ and finding short paths to connect $\beta(p)$ with $\beta(p_{\text{step}})$ in $MOP$
  - Controlling term size in subgoals (heuristics for expanding function definitions vs. treating them as uninterpreted)
- Fast and flexible SMT solvers
Selection of Related Work

- Burch & Dill [CAV 94]
- Damm & Pnueli [CHARME 97]
- Shen & Arvind [Formal Techn. for Hardware 98]
- Skakkebæk & al. [CAV 98]
- Hosabettu & al. [CAV 00]
- Lahiri & Bryant [CAV 03]
- Manolios & Srinivasan [ICCAD 05]