SCIENCE
PASSION
TECHNOLOGY

Benchmarking the Core, not the Compiler: LLVM Scheduling for the CV32E40P

Lorenz Schumm (Graz University of Technology)

24.02.2026

› www.isec.tugraz.at

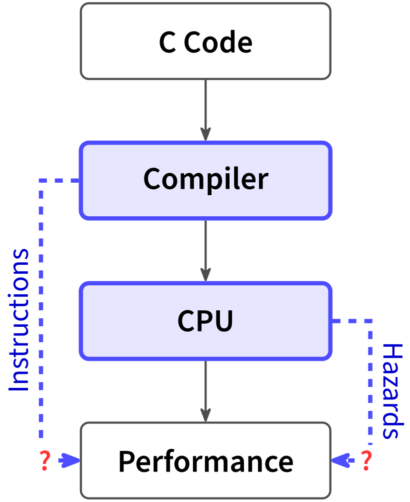
Summary

On **in-order** cores, runtime **performance** can be strongly influenced by compiler **instruction ordering**.

Motivation

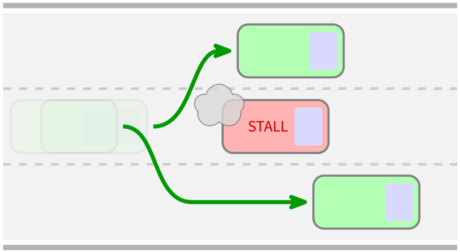
- **Context:** Extending CV32E40P
  - Pipeline modifications
  - Custom instructions
  - LLVM integration
- Inconsistent benchmark results

? *“Is the extension slow, or the compiler?”*

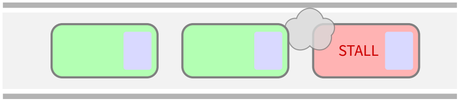


In-Order Core Stalls

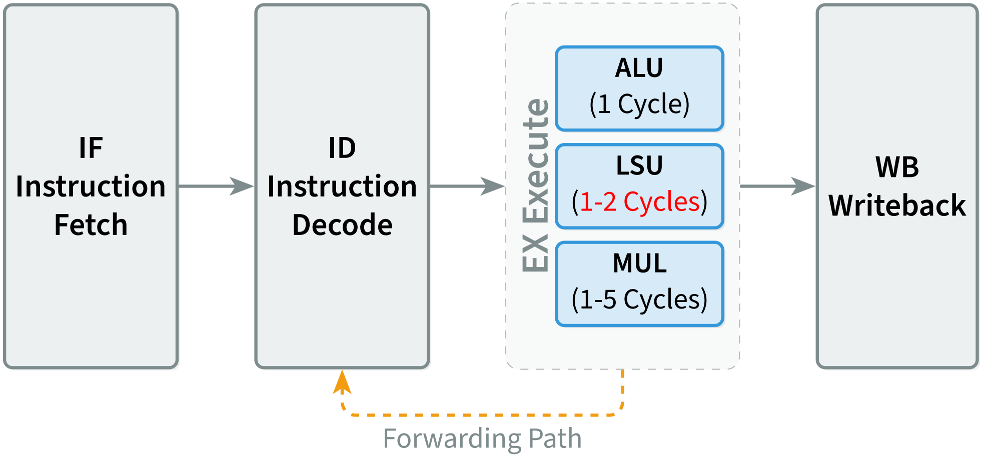
Out-of-Order (OoO)



In-Order (CV32E40P)



CV32E40P Pipeline



Load-Use Hazard

	C1	C2	C3	C4	C5	C6	C7
`LW a0, 0(sp)`	IF	ID	EX	WB			
`ADD a1, a0, a2`		IF	ID	STALL	EX	WB	
`SUB t0, t1, t2`			IF	STALL	ID	EX	WB

LLVM Instruction Scheduler

- Scheduling is **Built-in**, we add new CPU models and instructions.
- **Goal:** Reorder instructions to fill stalls without modifying logic.
- **LLVM TableGen:** Processor-specific modeling
  - Defines *Resources* and *Latencies*.

Result: Pipeline-Aware Compiler

- *“Load results are available after 2 cycles”*

Preventing Stalls

Baseline $\rightarrow$ **Scheduled**

Baseline

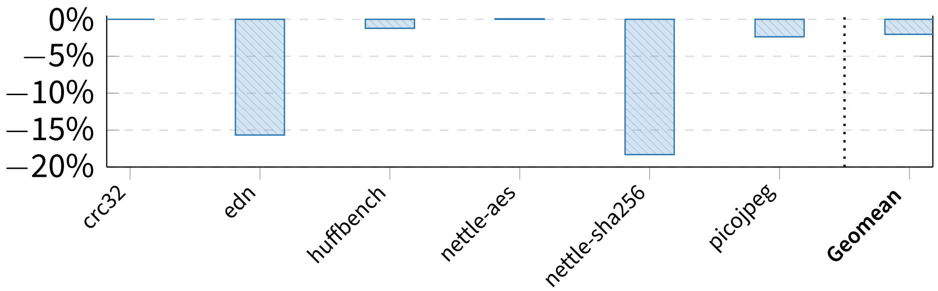
```
1 lw a0, 0(sp)
2 <Stall>
3 add a1, a0, a2
4 sub t0, t1, t2
```

Scheduled

```
1 lw a0, 0(sp)
2 sub t0, t1, t2
3 add a1, a0, a2
```

Performance Evaluation

Runtime reduction for subset of Embench-IoT suite:



Up to **18%** runtime reduction
(Zero C code modifications)

LLVM Scheduling Model

- Scheduling model is relatively easy to extend for pipeline changes or new instructions

TableGen Model Snippet

```
let BufferSize = 0 in {
  def CV32E40PUnitLSU  : ProcResource<1>;
}
def : WriteRes<WriteSTW, [CV32E40PUnitLSU]>; // fire-and-forget
let Latency = 2 in { // 1 extra cycle compared to e.g. ALU
    def : WriteRes<WriteLDW, [CV32E40PUnitLSU]>;
}
// defining instructions:
def SW : Store_rri<0b010, "sw">, Sched<[WriteSTW]>;
def LW : Load_ri<0b010, "lw">, Sched<[WriteLDW]>;
```

Related

- **Bottom-Up scheduling:** **Shorter value lifetimes** $\rightarrow$ smaller stack frames
- **Function alignment:** CV32E40P stalls on non-aligned calls
- **Loop alignment:** CV32E40P stalls on non-aligned jumps

Takeaway

- Compiler artifacts significantly influence in-order core benchmarks
- Even with **-O3**, artifacts remain
- **Pipeline modeling** in LLVM eliminates noise
- **Built-in LLVM feature**: Requires only CPU models/new instructions
- **Zero impact on functionality or source code**

$\rightarrow$ **Improved performance** for in-order cores
$\rightarrow$ More **accurate runtime benchmarking**

Conclusion

Thank you for your attention!

For students: check out the course *Secure System Architectures*
`https://isec.tugraz.at/ssa` (no SystemVerilog involved)