MINES Saint-Étienne

Institut Mines-Télécom

THALES
Building a future we can all trust

Post-Quantum Cryptography (PQC) Accelerated by a Superscalar RISC-V Processor

Thesis advised by: Jean-Roch Coulon, André Sintzoff, Olivier Potin and Jean-Baptiste Rigaud

Côme Allart
Thales DIS & Mines Saint-Étienne
Feb. 22nd, 2026

INSPIRING INNOVATION SINCE 1816

Introduction

Context

CIFRE PhD at Thales DIS: **embedded** products for digital identity and security

- Smart cards (ID card, access card, SIM card, bank card, etc.)
- Passports
- ...

Introduction

Context

CIFRE PhD at Thales DIS: **embedded** products for digital identity and security

- Smart cards (ID card, access card, SIM card, bank card, etc.)
- Passports
- ...

Secured products, digital vaults

- Need for cryptography
- Constrained execution time (EMV standard)
- Need for performance

Introduction

Context

CIFRE PhD at Thales DIS: **embedded** products for digital identity and security

- Smart cards (ID card, access card, SIM card, bank card, etc.)
- Passports
- ...

Secured products, digital vaults

- Need for cryptography
- Constrained execution time (EMV standard)
- Need for performance

Need for sovereign products development

Post-Quantum Cryptography Accelerated by a Superscalar RISC-V Processor

Post-Quantum Cryptography Accelerated by a Superscalar RISC-V Processor

Cryptography

- Asymmetric Cryptography (e.g. RSA) would be broken by a quantum computer

Post-Quantum Cryptography Accelerated by a Superscalar RISC-V Processor

Post-Quantum Cryptography (PQC)

- Asymmetric Cryptography (e.g. RSA) would be broken by a quantum computer
- Post-Quantum Cryptography is safe against both quantum and classic computers
- Complex algorithms that need hardware acceleration for performance

Post-Quantum Cryptography Accelerated by a Superscalar RISC-V Processor

Post-Quantum Cryptography (PQC)

- Asymmetric Cryptography (e.g. RSA) would be broken by a quantum computer
- Post-Quantum Cryptography is safe against both quantum and classic computers
- Complex algorithms that need hardware acceleration for performance

RISC-V Processor

- Processor: electronic device executing a program composed of instructions
- RISC-V: open and extensible Instruction Set Architecture (ISA)
  - **Open** to create processors implementing these instructions
  - **Extensible** to create new instructions

Post-Quantum Cryptography Accelerated by a Superscalar RISC-V Processor

Post-Quantum Cryptography (PQC)

- Asymmetric Cryptography (e.g. RSA) would be broken by a quantum computer
- Post-Quantum Cryptography is safe against both quantum and classic computers
- Complex algorithms that need hardware acceleration for **performance**

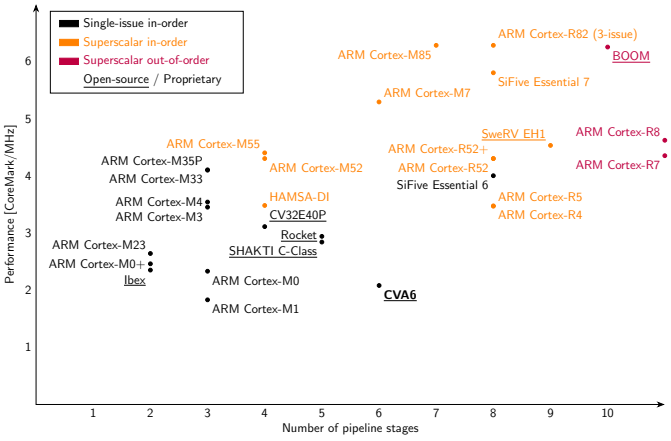
RISC-V Processor

- Processor: electronic device executing a program composed of instructions
- RISC-V: open and extensible Instruction Set Architecture (ISA)
  - **Open** to create processors implementing these instructions
  - **Extensible** to create new instructions

Superscalar: executes several instructions each clock cycle for more **performance**

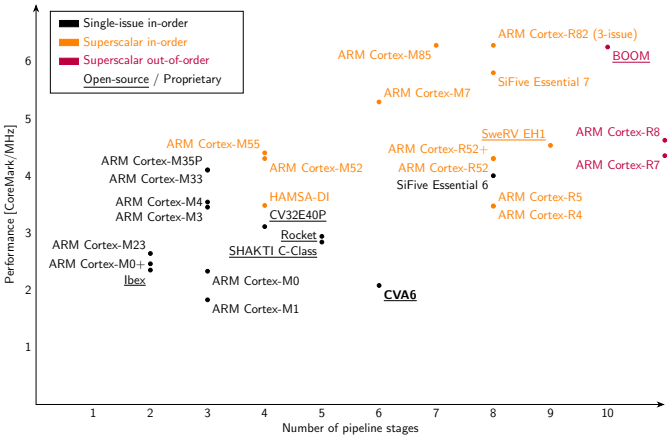
Introduction

Processor Choice



Introduction

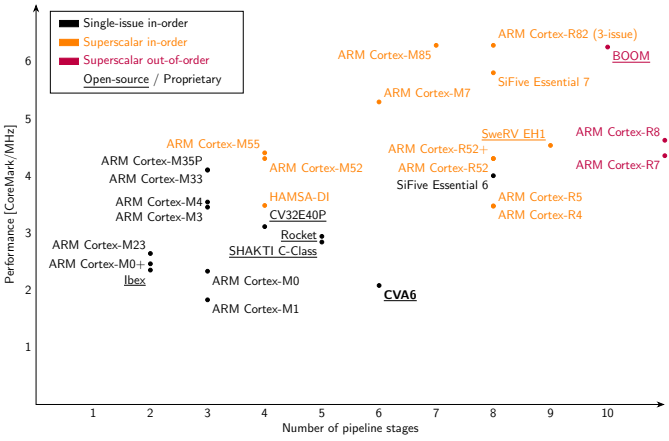
Processor Choice



Open-source processors

Introduction

Processor Choice

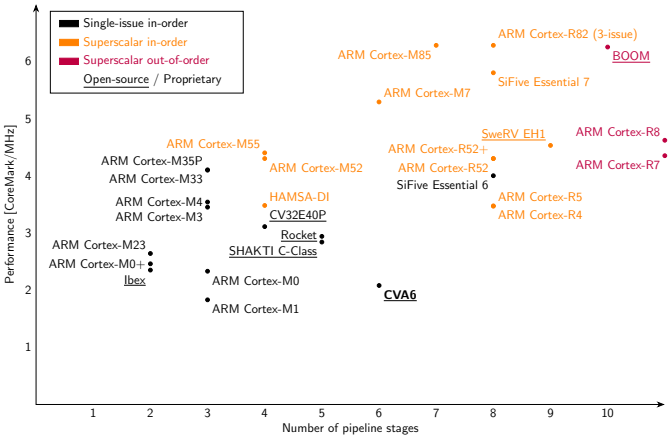


Open-source processors

- CV32E40P
- CVA6

Introduction

Processor Choice



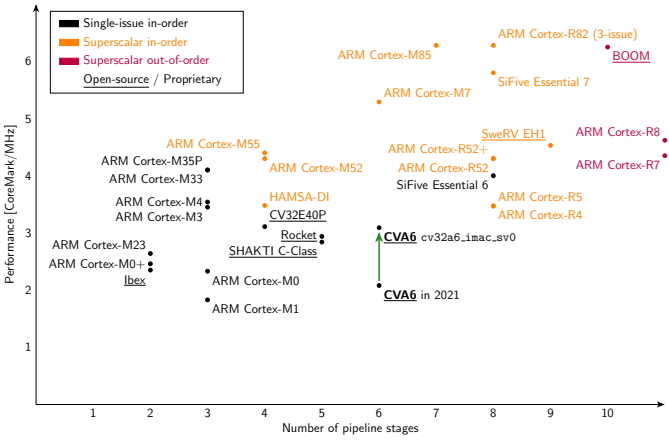
Open-source processors

- CV32E40P
- CVA6

CVA6 : 2.08 CoreMark/MHz

Introduction

Processor Choice



Open-source processors

- CV32E40P
- CVA6

CVA6 : **3.09 CoreMark/MHz**

Introduction

Application: the CVA6 processor

- Open-source RISC-V processor
- Created by F. Zaruba at ETH zürich (2019) and maintained by OPENHW FOUNDATION
- Mature for industry

Introduction

Application: the CVA6 processor

- Open-source RISC-V processor
- Created by F. Zaruba at ETH zürich (2019) and maintained by OPENHW FOUNDATION
- Mature for industry

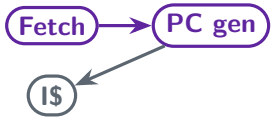
- *Register-Transfer Level* (RTL) model described in the SystemVerilog language
- More than 60 configuration options
  - 32- or 64-bit registers
  - Optional support of various RISC-V extensions

    A, B, C, F, H, V, Zcb, Zcmp, Zicond, modes S et U
- Performance options
  - Branch predictors
  - Scoreboard size
  - Etc.

- 6-stage pipeline

- 6-stage pipeline

- 6-stage pipeline



Introduction
CVA6 Pipeline

- 6-stage pipeline

- 6-stage pipeline
- **In-order operand reading**

- 6-stage pipeline
- **In-order operand reading**

Introduction

CVA6 Pipeline

- 6-stage pipeline
- In-order operand reading
- **Out-of-order execution**

Introduction

CVA6 Pipeline

- 6-stage pipeline
- In-order operand reading
- Out-of-order execution

Introduction

CVA6 Pipeline

- 6-stage pipeline
- In-order operand reading
- Out-of-order execution
- **Single-issue**

- At the beginning of the PhD: 3.09 CoreMark/MHz (GCC 13.1.0)

- At the beginning of the PhD: 3.09 CoreMark/MHz (GCC 13.1.0)
- Need for more performance: increase the CoreMark score

- At the beginning of the PhD: 3.09 CoreMark/MHz (GCC 13.1.0)
- Need for more performance: increase the CoreMark score
- A superscalar processor would improve **performance**

- At the beginning of the PhD: 3.09 CoreMark/MHz (GCC 13.1.0)
- Need for more performance: increase the CoreMark score
- A superscalar processor would improve **performance**
- But it would require **complex changes**

- At the beginning of the PhD: 3.09 CoreMark/MHz (GCC 13.1.0)
- Need for more performance: increase the CoreMark score
- A superscalar processor would improve **performance**
- But it would require **complex changes**

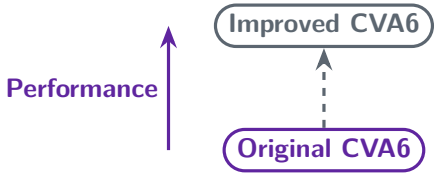
Previous attempt at making CVA6 superscalar (GSoC 2019): only 16 % performance gain

- At the beginning of the PhD: 3.09 CoreMark/MHz (GCC 13.1.0)
- Need for more performance: increase the CoreMark score
- A superscalar processor would improve **performance**
- But it would require **complex changes**

Previous attempt at making CVA6 superscalar (GSoC 2019): only 16 % performance gain

How to make sure that superscalar brings a performance gain before performing deep changes inside CVA6?

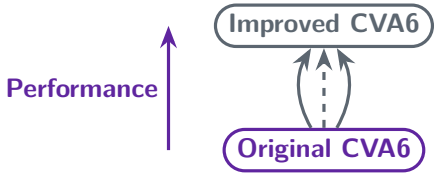
Goal: **improve CVA6 performance, especially for PQC**



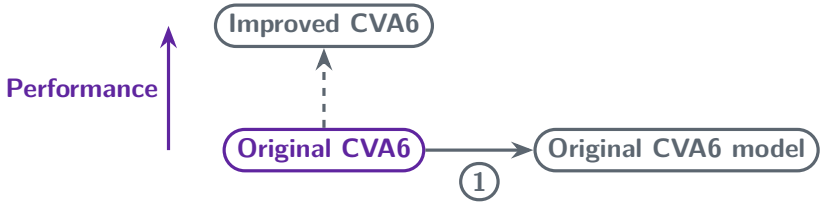
Introduction

Method: model-guided design

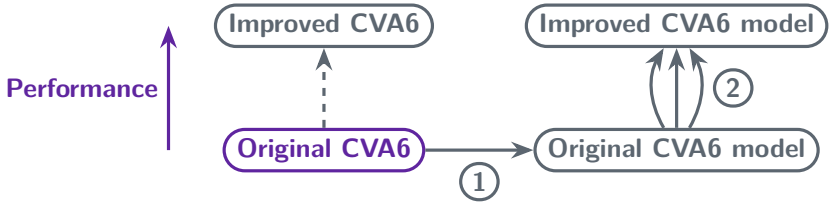
Goal: **improve CVA6 performance, especially for PQC**



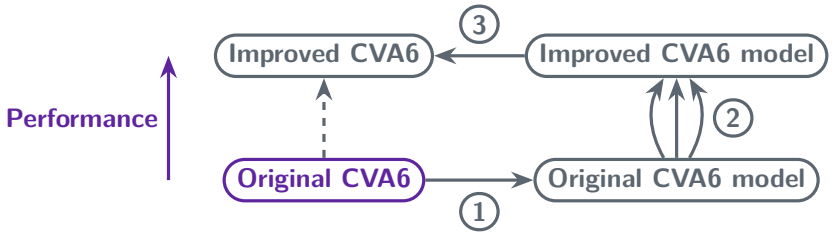
Goal: **improve CVA6 performance, especially for PQC**



Goal: **improve CVA6 performance, especially for PQC**



Goal: **improve CVA6 performance, especially for PQC**

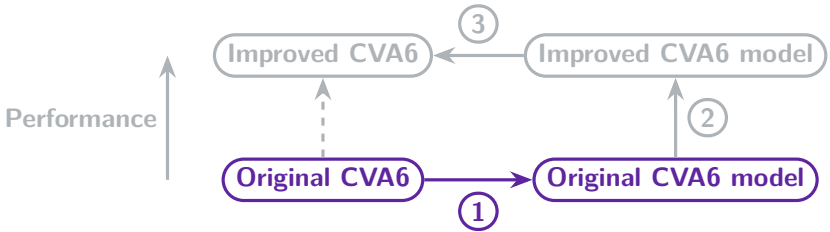


Plan

CVA6 Performance Modeling

CVA6 Performance Modeling

Model creation



1. **Easy to modify**
   - May be slow to run

CVA6 Performance Modeling

Requirements

1. **Easy to modify**
   - May be slow to run
2. Detailed modeling of **control path**
   - Has to be **cycle-accurate**
   - Data path (behavior) not needed

CVA6 Performance Modeling

Requirements

1. **Easy to modify**
   - May be slow to run
2. Detailed modeling of **control path**
   - Has to be **cycle-accurate**
   - Data path (behavior) not needed
3. **Accurate**

CVA6 Performance Modeling

Requirements

1. **Easy to modify**
   - May be slow to run
2. Detailed modeling of **control path**
   - Has to be **cycle-accurate**
   - Data path (behavior) not needed
3. **Accurate**

- Did not use Gem5

CVA6 Performance Modeling

Requirements

1. **Easy to modify**
   - May be slow to run
2. Detailed modeling of **control path**
   - Has to be **cycle-accurate**
   - Data path (behavior) not needed
3. **Accurate**

- Did not use Gem5
- Did not use mechanistic or empirical modeling

CVA6 Performance Modeling

Requirements

1. **Easy to modify**
   - May be slow to run
2. Detailed modeling of **control path**
   - Has to be **cycle-accurate**
   - Data path (behavior) not needed
3. **Accurate**

- Did not use Gem5
- Did not use mechanistic or empirical modeling
- Chose Python

CVA6 Performance Modeling

Requirements

1. **Easy to modify**
   - May be slow to run
2. Detailed modeling of **control path**
   - Has to be **cycle-accurate**
   - Data path (behavior) not needed
3. **Accurate**

- Did not use Gem5
- Did not use mechanistic or empirical modeling
- Chose Python
- Constant-time operations
  - Data / addresses are ignored
  - **Cache misses are not modeled**

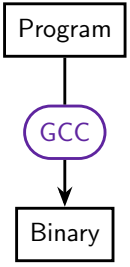
CVA6 Performance Modeling

Model Usage

Program

CVA6 Performance Modeling

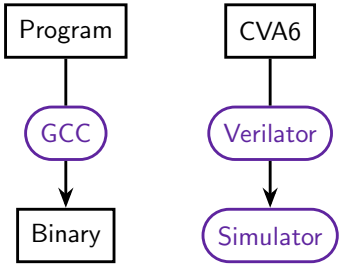
Model Usage




```
.LBB0_2:
    mul  a0, a2, a0
    addi a2, a2, 1
    bne  a1, a2, .LBB0_2
    ret
```

CVA6 Performance Modeling

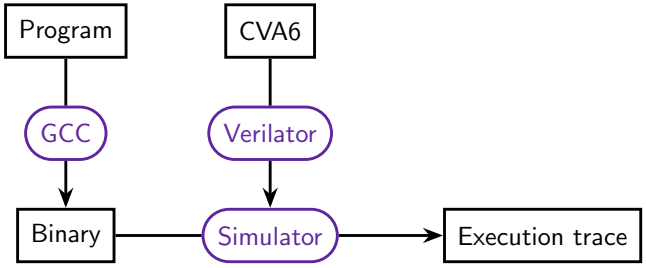
Model Usage




```
.LBB0_2:
    mul  a0, a2, a0
    addi a2, a2, 1
    bne  a1, a2, .LBB0_2
    ret
```

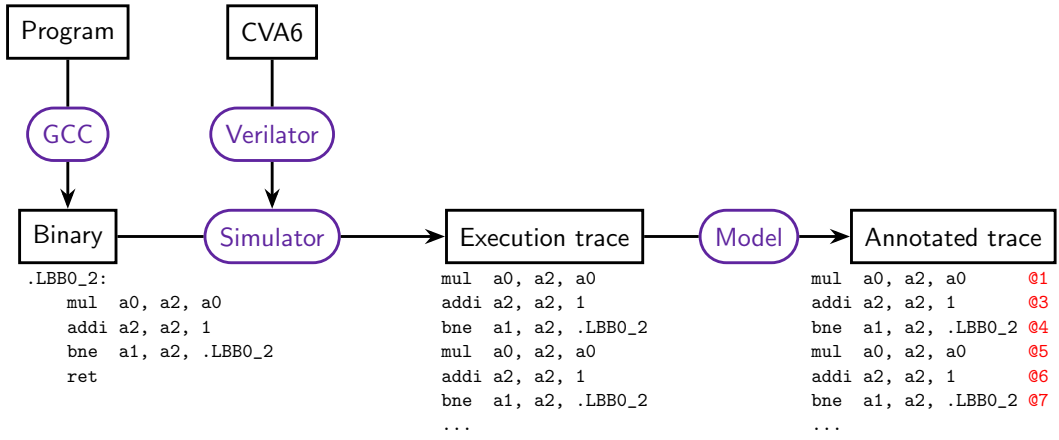
CVA6 Performance Modeling

Model Usage



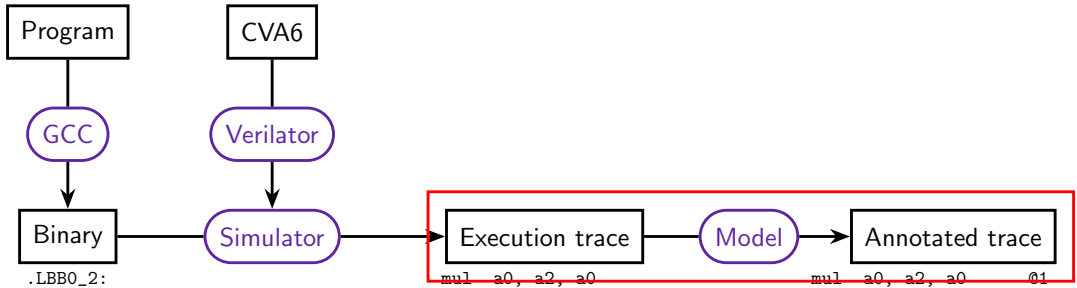
CVA6 Performance Modeling

Model Usage



CVA6 Performance Modeling

Model Usage



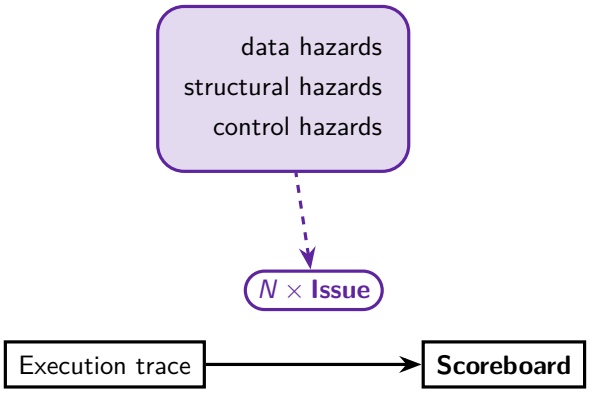
CVA6 Performance Modeling

Architecture: data and functions

Execution trace

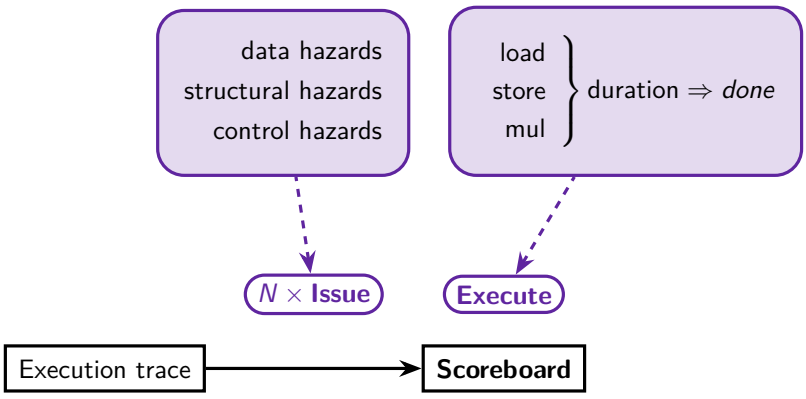
CVA6 Performance Modeling

Architecture: data and functions



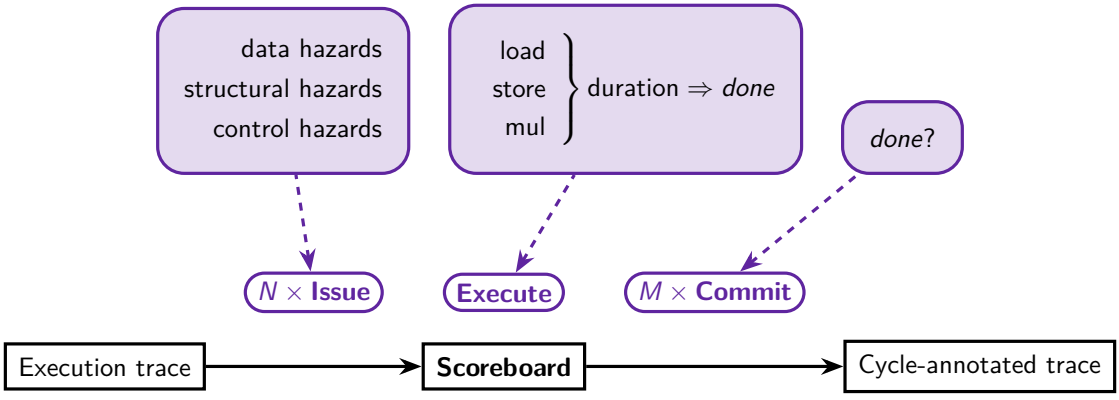
CVA6 Performance Modeling

Architecture: data and functions



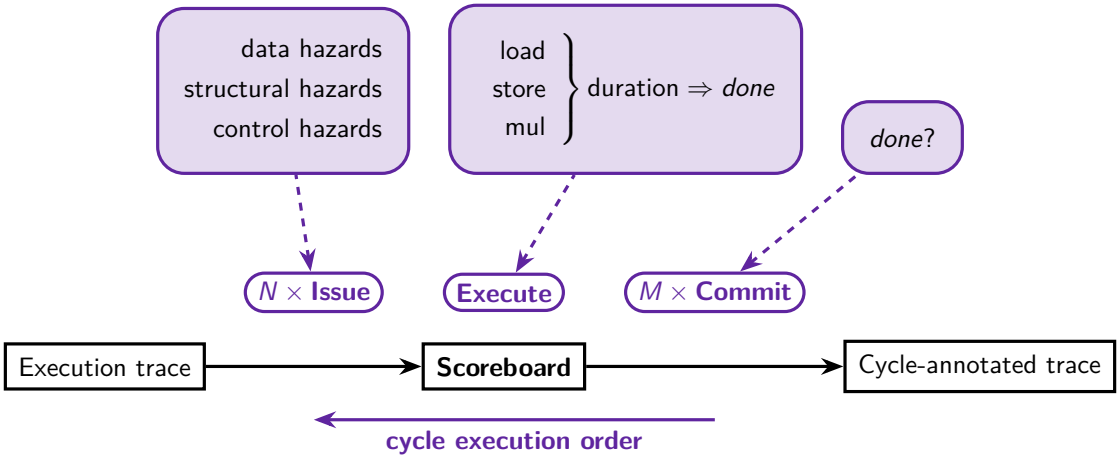
CVA6 Performance Modeling

Architecture: data and functions



CVA6 Performance Modeling

Architecture: data and functions



CVA6 Performance Modeling

Accuracy measurement

- t_i = commit cycle of i^{th} instruction

i	t_i^{Model}	t_i^{RTL}
$\vdots$	10	14
`add`	11	15
`mul`	12	17
`add`	13	18

CVA6 Performance Modeling

Accuracy measurement

- t_i = commit cycle of i^{th} instruction
- $\Delta t_i = t_i - t_{i-1}$

i	t_i^{Model}	t_i^{RTL}	$\Delta t_i^{\text{Model}}$	Δt_i^{RTL}
⋮	10	14	⋮	⋮
`add`	11	15	1	1
`mul`	12	17	1	2
`add`	13	18	1	1

CVA6 Performance Modeling

Accuracy measurement

- t_i = commit cycle of i^{th} instruction
- $\Delta t_i = t_i - t_{i-1}$
- $\Delta t_i^{\text{Model}}$ vs. Δt_i^{RTL}

i	t_i^{Model}	t_i^{RTL}	$\Delta t_i^{\text{Model}}$	Δt_i^{RTL}	
⋮	10	14	⋮	⋮	⋮
`add`	11	15	1	1	$=$
`mul`	12	17	1	2	$\neq$
`add`	13	18	1	1	$=$

CVA6 Performance Modeling

Accuracy measurement

- t_i = commit cycle of i^{th} instruction
- $\Delta t_i = t_i - t_{i-1}$
- $\Delta t_i^{\text{Model}}$ vs. Δt_i^{RTL}
- $\#\{i \mid \Delta t_i^{\text{Model}} = \Delta t_i^{\text{RTL}}\}$

i	t_i^{Model}	t_i^{RTL}	$\Delta t_i^{\text{Model}}$	Δt_i^{RTL}		=
⋮	10	14	⋮	⋮	⋮	
`add`	11	15	1	1	=	1
`mul`	12	17	1	2	≠	
`add`	13	18	1	1	=	**2**

CVA6 Performance Modeling

Accuracy measurement

- t_i = commit cycle of i^{th} instruction
- $\Delta t_i = t_i - t_{i-1}$
- $\Delta t_i^{\text{Model}}$ vs. Δt_i^{RTL}
- $\#\{i \mid \Delta t_i^{\text{Model}} = \Delta t_i^{\text{RTL}}\}$
- $\#\{i\}$

i	t_i^{Model}	t_i^{RTL}	$\Delta t_i^{\text{Model}}$	Δt_i^{RTL}		$=$	$\#$
$\vdots$	10	14	$\vdots$	$\vdots$	$\vdots$		
`add`	11	15	1	1	$=$	1	1
`mul`	12	17	1	2	$\neq$		2
`add`	13	18	1	1	$=$	**2**	**3**

CVA6 Performance Modeling

Accuracy measurement

- t_i = commit cycle of i^{th} instruction
- $\Delta t_i = t_i - t_{i-1}$
- $\Delta t_i^{\text{Model}}$ vs. Δt_i^{RTL}
- $\#\{i \mid \Delta t_i^{\text{Model}} = \Delta t_i^{\text{RTL}}\}$
- $\#\{i\}$

i	t_i^{Model}	t_i^{RTL}	$\Delta t_i^{\text{Model}}$	Δt_i^{RTL}		=	#
⋮	10	14	⋮	⋮	⋮		
`add`	11	15	1	1	=	1	1
`mul`	12	17	1	2	≠		2
`add`	13	18	1	1	=	**2**	**3**

$$\text{Accuracy} = \frac{\#\{i \mid \Delta t_i^{\text{Model}} = \Delta t_i^{\text{RTL}}\}}{\#\{i\}}$$

$$\text{Accuracy} = 2/3 = 67\,\%$$

CVA6 Performance Modeling

Accuracy measurement

- t_i = commit cycle of i^{th} instruction
- $\Delta t_i = t_i - t_{i-1}$
- $\Delta t_i^{\text{Model}}$ vs. Δt_i^{RTL}
- $\#\{i \mid \Delta t_i^{\text{Model}} = \Delta t_i^{\text{RTL}}\}$
- $\#\{i\}$

i	t_i^{Model}	t_i^{RTL}	$\Delta t_i^{\text{Model}}$	Δt_i^{RTL}		=	#
⋮	10	14	⋮	⋮	⋮		
`add`	11	15	1	1	=	1	1
`mul`	13	17	2	2	=	2	2
`add`	14	18	1	1	=	**3**	**3**

$$\text{Accuracy} = \frac{\#\{i \mid \Delta t_i^{\text{Model}} = \Delta t_i^{\text{RTL}}\}}{\#\{i\}}$$

$$\text{Accuracy} = 3/3 = 100\,\%$$

CVA6 Performance Modeling

Accuracy measurement

- t_i = commit cycle of i^{th} instruction
- $\Delta t_i = t_i - t_{i-1}$
- $\Delta t_i^{\text{Model}}$ vs. Δt_i^{RTL}
- $\#\{i \mid \Delta t_i^{\text{Model}} = \Delta t_i^{\text{RTL}}\}$
- $\#\{i\}$

i	t_i^{Model}	t_i^{RTL}	$\Delta t_i^{\text{Model}}$	Δt_i^{RTL}		=	#
⋮	10	14	⋮	⋮	⋮		
`add`	11	15	1	1	=	1	1
`mul`	13	17	2	2	=	2	2
`add`	14	18	1	1	=	**3**	**3**

$$\text{Accuracy} = \frac{\#\{i \mid \Delta t_i^{\text{Model}} = \Delta t_i^{\text{RTL}}\}}{\#\{i\}}$$

$$\text{Accuracy} = 3/3 = 100\,\%$$

99.2 % on CoreMark's 2nd iteration
(266,000 dynamic instructions)

CVA6 Performance Modeling

Accuracy measurement

- t_i = commit cycle of i^{th} instruction
- $\Delta t_i = t_i - t_{i-1}$
- $\Delta t_i^{\text{Model}}$ vs. Δt_i^{RTL}
- $\#\{i \mid \Delta t_i^{\text{Model}} = \Delta t_i^{\text{RTL}}\}$
- $\#\{i\}$

i	t_i^{Model}	t_i^{RTL}	$\Delta t_i^{\text{Model}}$	Δt_i^{RTL}		=	#
⋮	10	14	⋮	⋮	⋮		
`add`	11	15	1	1	=	1	1
`mul`	13	17	2	2	=	2	2
`add`	14	18	1	1	=	**3**	**3**

$$\text{Accuracy} = \frac{\#\{i \mid \Delta t_i^{\text{Model}} = \Delta t_i^{\text{RTL}}\}}{\#\{i\}}$$

$$\text{Accuracy} = 3/3 = 100\,\%$$

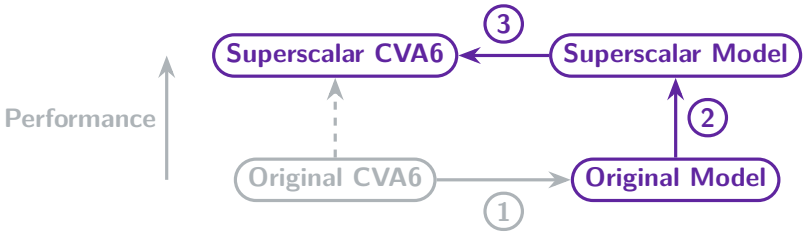
99.2 % on CoreMark's 2nd iteration
(266,000 dynamic instructions)

Allart et al. *"Performance Modeling of CVA6 with Cycle-Based Simulation"*. RISC-V Summit Europe. 2023

Superscalar CVA6 Development

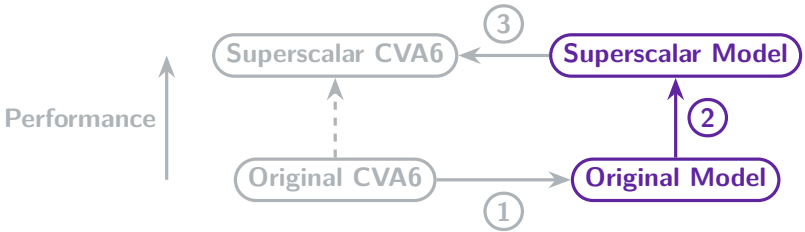
Superscalar CVA6 Development

Methodology Reminder



Superscalar CVA6 Development

Modeling Phase



Pipeline = several instruction *parts* in parallel

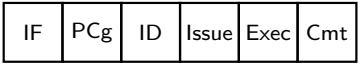
$\neq$

Superscalar = several instructions in parallel

Pipeline = several instruction *parts* in parallel

$\neq$

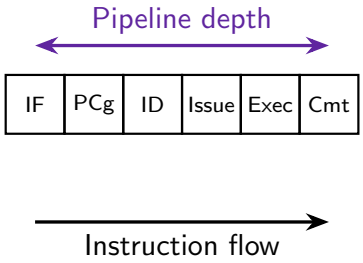
Superscalar = several instructions in parallel



Pipeline = several instruction *parts* in parallel

$\neq$

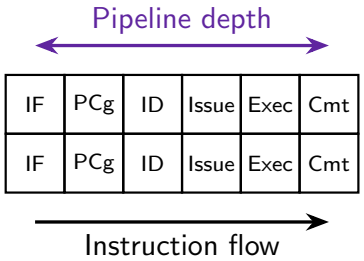
Superscalar = several instructions in parallel



Pipeline = several instruction *parts* in parallel

$\neq$

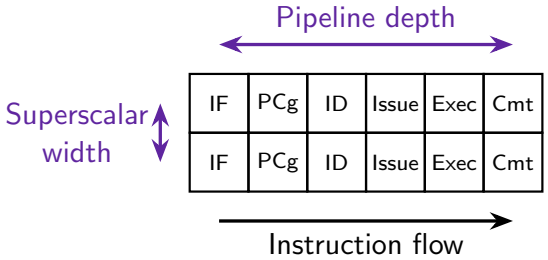
Superscalar = several instructions in parallel



Pipeline = several instruction *parts* in parallel

$\neq$

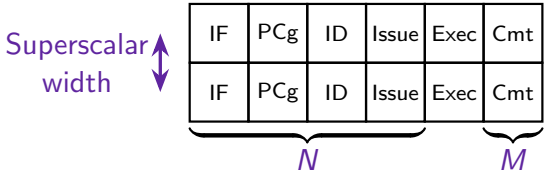
Superscalar = several instructions in parallel



Pipeline = several instruction *parts* in parallel

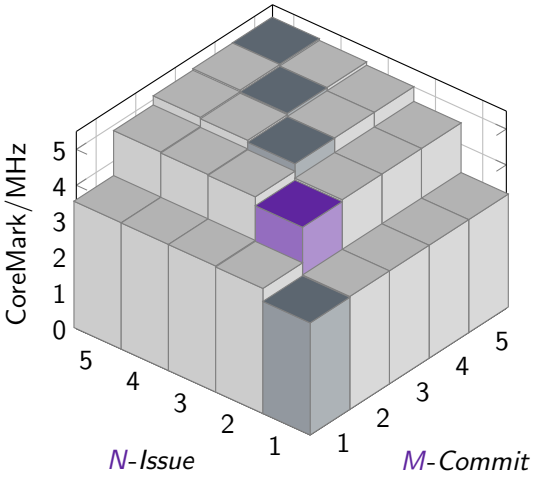
$\neq$

Superscalar = several instructions in parallel



Superscalar CVA6 Development

Modeling Phase: Results

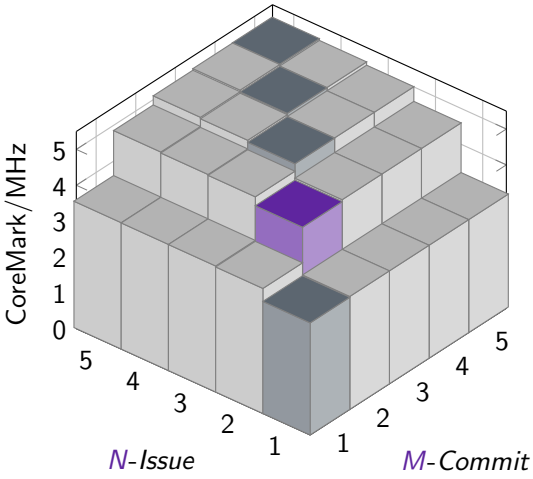


Superscalar width

- Chose $N{=}M{=}2$
- Estimated 46 % performance gain: 4.42 CoreMark/MHz
- Area cost supposedly lower than for $N{=}M{=}3$

Superscalar CVA6 Development

Modeling Phase: Results



Superscalar width

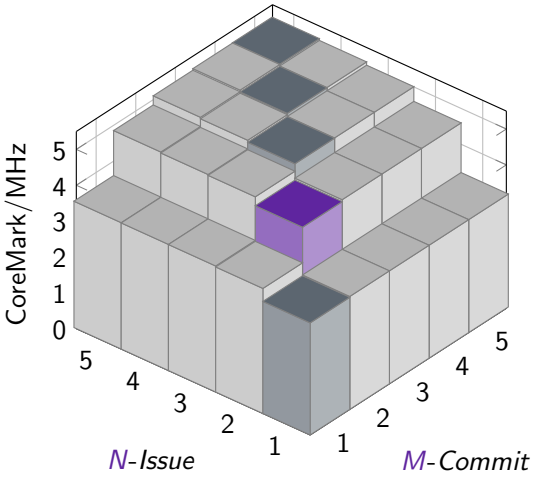
- Chose $N{=}M{=}2$
- Estimated 46 % performance gain: 4.42 CoreMark/MHz
- Area cost supposedly lower than for $N{=}M{=}3$

Execute stage architecture

- Add an additional ALU
- ALU assignment precedence

Superscalar CVA6 Development

Modeling Phase: Results



Superscalar width

- Chose $N{=}M{=}2$
- Estimated 46 % performance gain: 4.42 CoreMark/MHz
- Area cost supposedly lower than for $N{=}M{=}3$

Execute stage architecture

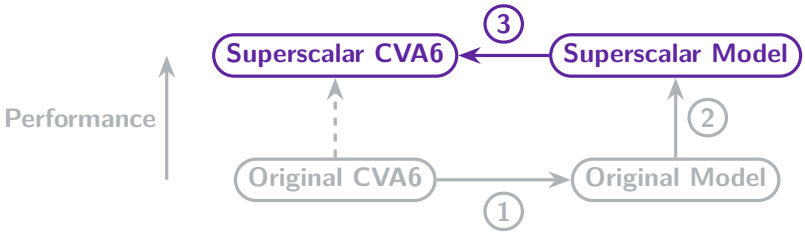
- Add an additional ALU
- ALU assignment precedence

Other microarchitectural choices

- Scoreboard size
- Instruction fetch bus width
- Etc.

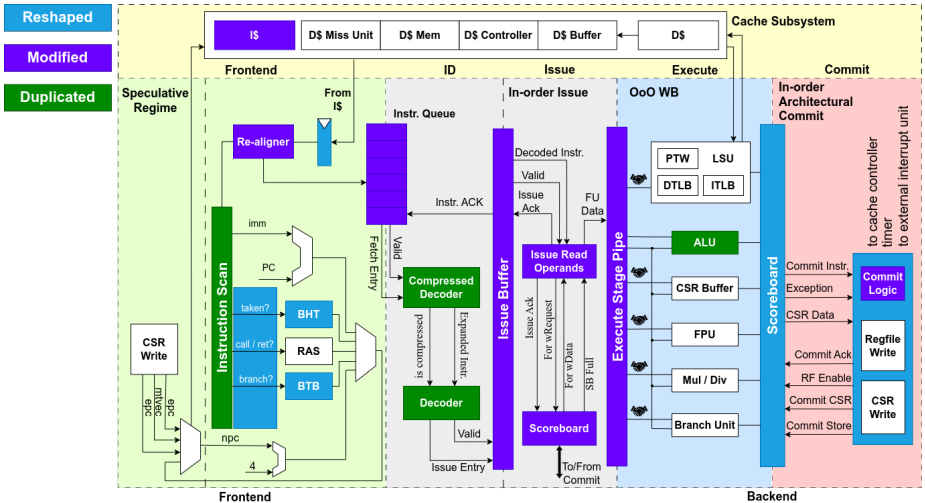
Superscalar CVA6 Development

Development Phase



Superscalar CVA6 Development

Development Phase



Comparison with the model performed regularly during development

(α) Global performance on the whole benchmark

- Performance gain indicated by the benchmark

Comparison with the model performed regularly during development

(α) Global performance on the whole benchmark
- Performance gain indicated by the benchmark

(β) Local performance on instruction sequences
- Annotated trace with the duration of each instruction

Superscalar CVA6 Development

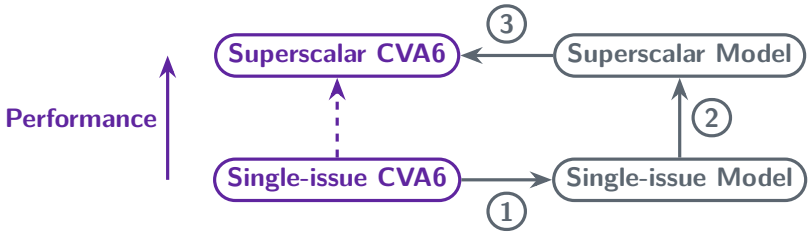
Using the Model as a Reference

Comparison with the model performed regularly during development

- (α) Global performance on the whole benchmark
  - Performance gain indicated by the benchmark
- (β) Local performance on instruction sequences
  - Annotated trace with the duration of each instruction
- (γ) Internal state and logic
  - The model logs its scoreboard state and decisions every cycle
  - These logs are compared with waveform from RTL simulation

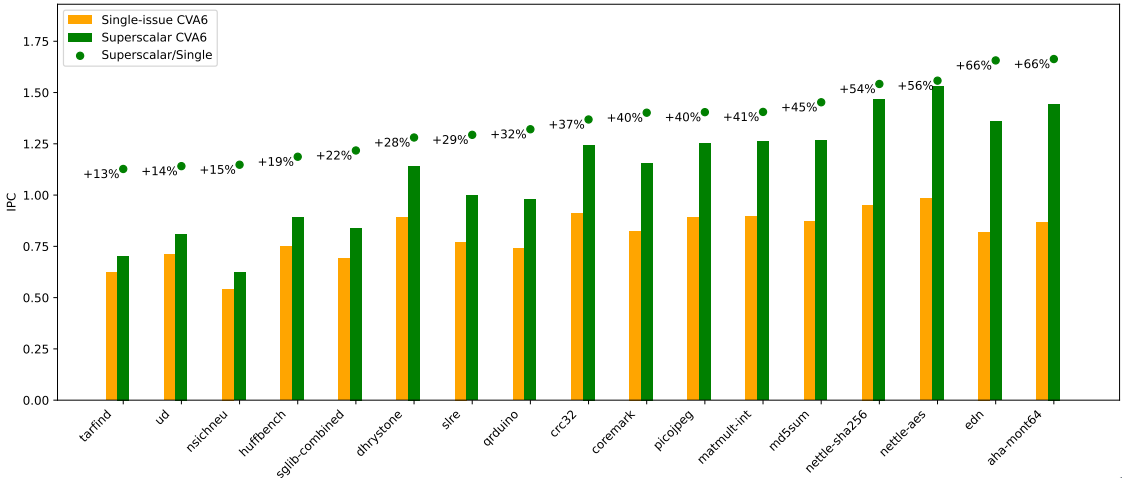
Superscalar CVA6 Development

Results



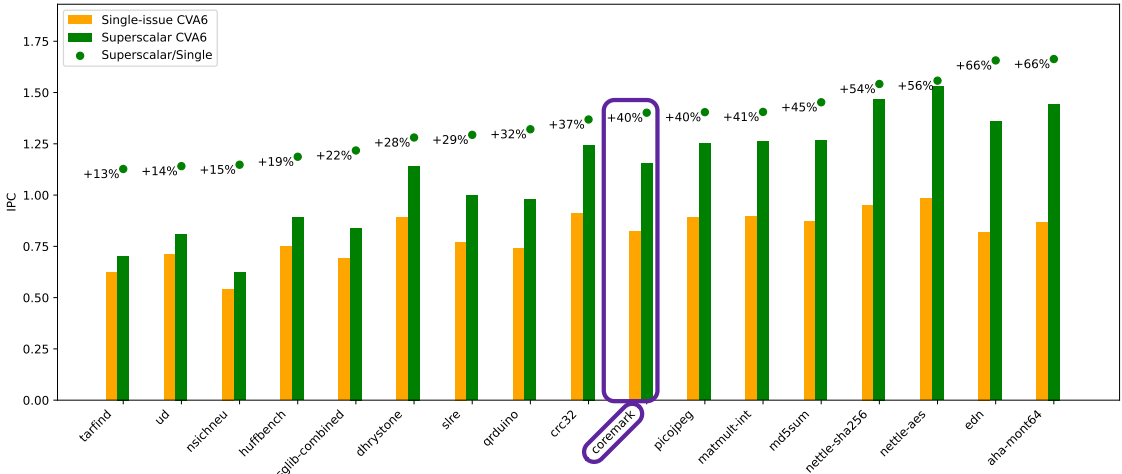
Superscalar CVA6 Development

Performance gain on CoreMark, Dhrystone and Embench



Superscalar CVA6 Development

Performance gain on CoreMark, Dhrystone and Embench



Superscalar CVA6 Development

Implementation Results

Performance gain vs single-issue

Target	Minimum	Average	Maximum
Superscalar (RTL simulation)	+13 %	**+36 %**	+66 %

Superscalar CVA6 Development

Implementation Results

Performance gain vs single-issue

Target	Minimum	Average	Maximum
Superscalar (RTL simulation)	+13 %	**+36 %**	+66 %

Superscalar CVA6 `cv32a65x`

- Open-source
- Optional with RTL configuration
- Passes all regression tests
- Runs **Linux** on an **FPGA**
- **4.35 CoreMark/MHz**
- Logic synthesis for **ASIC** technology

Superscalar CVA6 Development

Implementation Results

Performance gain vs single-issue			
Target	Minimum	Average	Maximum
Superscalar (RTL simulation)	+13 %	**+36 %**	+66 %

Superscalar CVA6 `cv32a65x`

- Open-source
- Optional with RTL configuration
- Passes all regression tests
- Runs **Linux** on an **FPGA**
- **4.35 CoreMark/MHz**
- Logic synthesis for **ASIC** technology

	Reference	Superscalar	Variation
Max. freq.	892 MHz	877 MHz	−1.7 %
Area	250 $kGates_{eq}$	278 $kGates_{eq}$	+11 %
Consumption*	32.5 mW	34.8 mW	+7.1 %

* : Rough estimation

Superscalar CVA6 Development

Implementation Results

Performance gain vs single-issue

Target	Minimum	Average	Maximum
Superscalar (RTL simulation)	+13 %	**+36 %**	+66 %

Superscalar CVA6 `cv32a65x`

- Open-source
- Optional with RTL configuration
- Passes all regression tests
- Runs **Linux** on an **FPGA**
- **4.35 CoreMark/MHz**
- Logic synthesis for **ASIC** technology

	Reference	Superscalar	Variation
Max. freq.	892 MHz	877 MHz	−1.7 %
Area	250 $kGates_{eq}$	278 $kGates_{eq}$	+11 %
Consumption*	32.5 mW	34.8 mW	+7.1 %

* : Rough estimation

Allart et al. *"Using a Performance Model to Implement a Superscalar CVA6"*.
21st ACM International Conference on Computing Frontiers. 2024

PQC in the Superscalar CVA6

PQC in the Superscalar CVA6

Context

Post-Quantum Cryptography: resists against both quantum and classic computers

The *National Institute of Standards and Technology* (NIST) ratified in 2024

- FIPS 203 : ML-KEM (Kyber) for key encapsulation and encryption
- FIPS 204 : ML-DSA (Dilithium) for digital signature
- FIPS 205 : SLH-DSA (SPHINCS+) as an alternative to ML-DSA

Application: **ML-DSA** (Dilithium)

ML-DSA consists of 3 algorithms

1. Key generation
2. Signature
3. Verification

PQC in the Superscalar CVA6

Goal

Accelerate Post-Quantum Cryptography (PQC)

ML-DSA executes 3 operation types

- Hash using Keccak (81 % for key generation according to PQ.V.ALU.E)
- Polynomial operations (multiplications) (12 %)
- Other operations (7 %)

These base functions are similar for ML-KEM.

PQC in the Superscalar CVA6

Goal

Accelerate Post-Quantum Cryptography (PQC)

ML-DSA executes 3 operation types

- Hash using Keccak (81 % for key generation according to PQ.V.ALU.E)
- Polynomial operations (multiplications) (12 %)
- Other operations (7 %)

These base functions are similar for ML-KEM.

Greatest performance budget : Keccak

- Best solution: acceleration outside the processor

PQC in the Superscalar CVA6

Goal

Accelerate Post-Quantum Cryptography (PQC)

ML-DSA executes 3 operation types

- Hash using Keccak (81 % for key generation according to PQ.V.ALU.E)
- Polynomial operations (multiplications) (12 %)
- Other operations (7 %)

These base functions are similar for ML-KEM.

Greatest performance budget : Keccak

- Best solution: acceleration outside the processor

Second performance budget is the **polynomial multiplication via NTT**

PQC in the Superscalar CVA6

Goal

Accelerate Post-Quantum Cryptography (PQC)

ML-DSA executes 3 operation types

- Hash using Keccak **accelerated** (81 $\rightarrow$ 3 % for key generation according to PQ.V.ALU.E)
- Polynomial operations (multiplications) (12 $\rightarrow$ 63 %)
- Other operations (7 $\rightarrow$ 34 %)

These base functions are similar for ML-KEM.

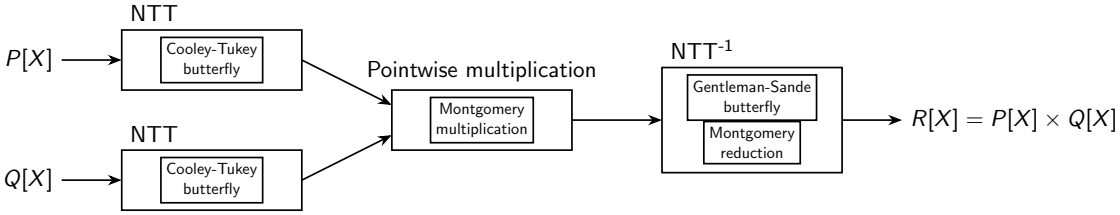
Greatest performance budget : Keccak

- Best solution: acceleration outside the processor

Second performance budget is the **polynomial multiplication via NTT**

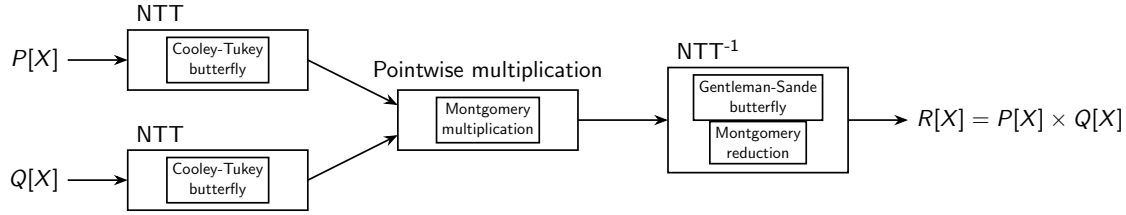
PQC in the Superscalar CVA6

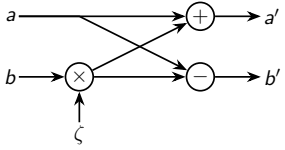
Polynomial Multiplication via NTT



PQC in the Superscalar CVA6

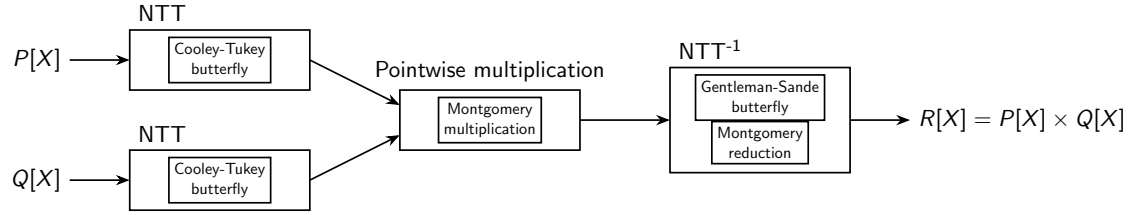
Polynomial Multiplication via NTT

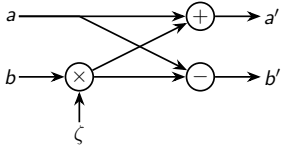


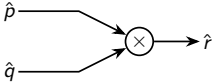


PQC in the Superscalar CVA6

Polynomial Multiplication via NTT

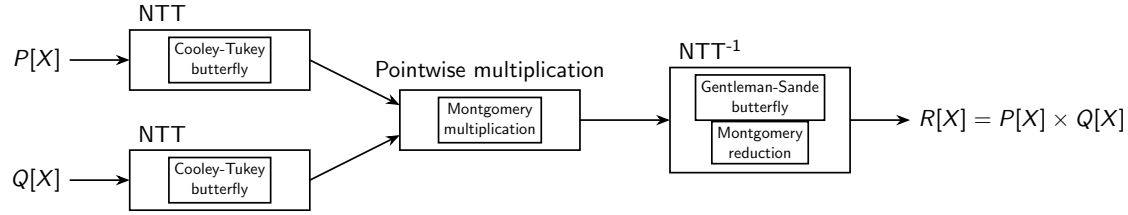


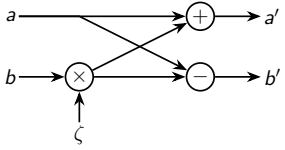


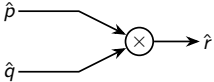


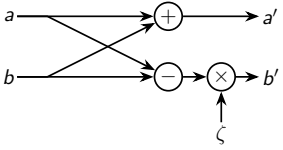
PQC in the Superscalar CVA6

Polynomial Multiplication via NTT



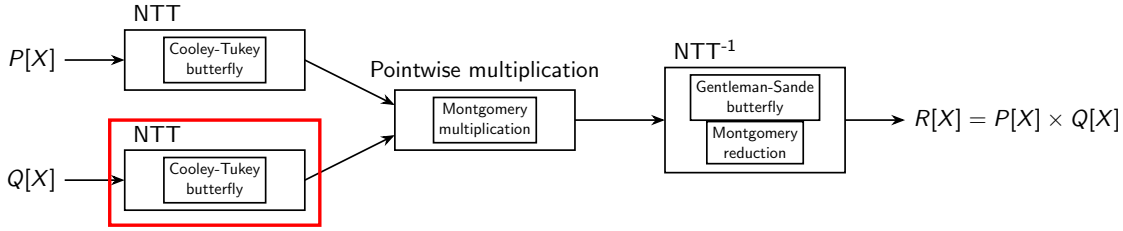


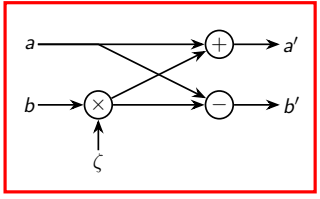


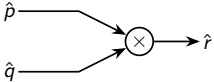


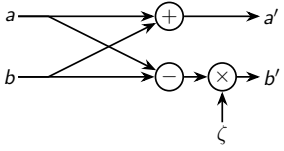
PQC in the Superscalar CVA6

Polynomial Multiplication via NTT



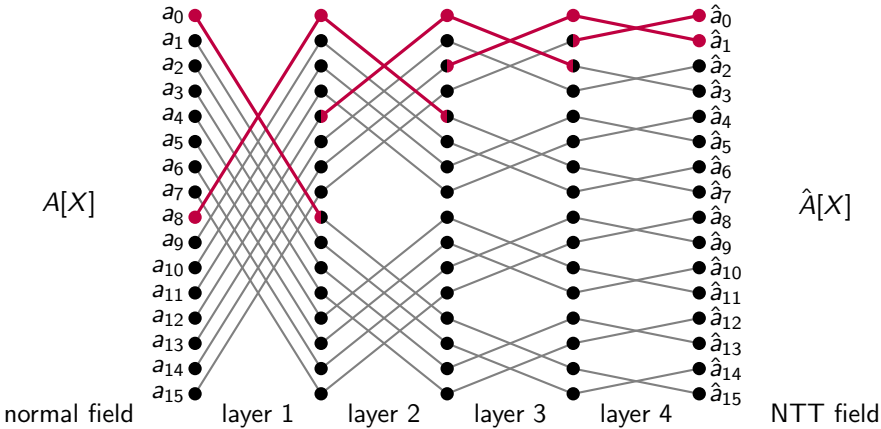






PQC in the Superscalar CVA6

Example of a 16-coefficient NTT



PQC in the Superscalar CVA6

State of the Art of NTT Hardware Acceleration

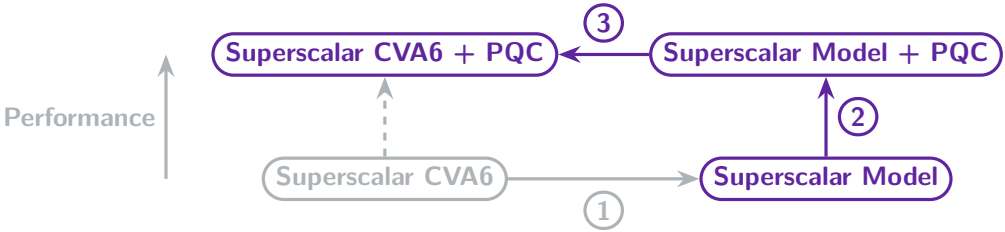
Duration of NTT, NTT^{-1} and pointwise multiplication (cycles)

Acceleration	Processor	Solution	NTT	NTT^{-1}	Pt.wise Mult.
Loosely-coupled Coprocessor	CVA6	CRYPHTOR	1074	1074	$264^{(1)}$
	CV32E40X	ATHOS	1531	1531	/
Inside the pipeline	RI5CY	32-bit RISC-V	17041	20372	4346
		PQ.V.ALU.E	**2705**	**3979**	**1274**
		Variation	-84 %	-80 %	-71 %
	Single-issue CVA6	64-bit RISC-V	38043	46266	/
		NTT Instructions	18554	21375	/
		Variation	-51 %	-54 %	/

Issue: how to handle the two output coefficients?

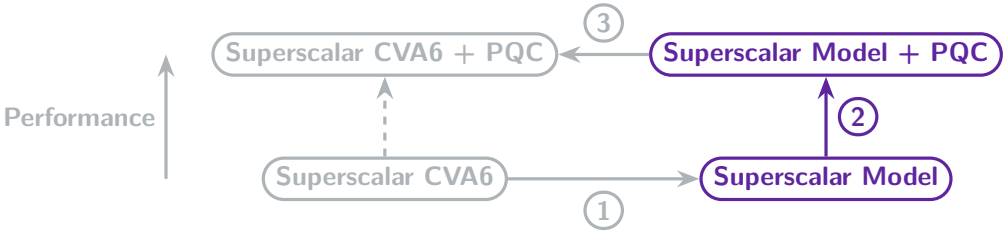
PQC in the Superscalar CVA6

Methodology Reminder



PQC in the Superscalar CVA6

Modeling Phase



PQC in the Superscalar CVA6

Modeling Phase: Methodology Extension

Issue

- An execution trace is required
- **However**, the butterfly instruction does not exist yet

PQC in the Superscalar CVA6

Modeling Phase: Methodology Extension

Issue

- An execution trace is required
- **However**, the butterfly instruction does not exist yet

Methodology extension: creating a new instruction

PQC in the Superscalar CVA6

Modeling Phase: Methodology Extension

Issue

- An execution trace is required
- **However**, the butterfly instruction does not exist yet

Methodology extension: creating a new instruction

1. Choose an existing **instruction** which is not used in the given program (e.g. `xor`)

Issue

- An execution trace is required
- **However**, the butterfly instruction does not exist yet

Methodology extension: creating a new instruction

1. Choose an existing **instruction** which is not used in the given program (e.g. `xor`)
2. In the C code, replace the butterfly function by the `xor` instruction

Issue

- An execution trace is required
- **However**, the butterfly instruction does not exist yet

Methodology extension: creating a new instruction

1. Choose an existing **instruction** which is not used in the given program (e.g. `xor`)
2. In the C code, replace the butterfly function by the `xor` instruction
3. In the model, replace the `xor` instructions by butterfly instructions

Issue

- An execution trace is required
- **However**, the butterfly instruction does not exist yet

Methodology extension: creating a new instruction

1. Choose an existing **instruction** which is not used in the given program (e.g. `xor`)
2. In the C code, replace the butterfly function by the `xor` instruction
3. In the model, replace the `xor` instructions by butterfly instructions
4. In the model body, model the **new butterfly instruction**

PQC in the Superscalar CVA6

Modeling Phase: Changes in the Model

Operations to perform

- Load
- Butterfly
  - $a' = a + z.b \mod q$
  - $b' = a - z.b \mod q$
- Store

PQC in the Superscalar CVA6

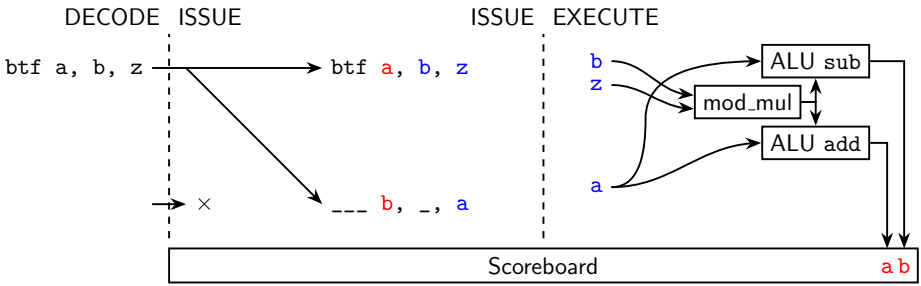
Modeling Phase: Changes in the Model

Operations to perform

- Load
- **Butterfly**
  - $a' = a + z.b \mod q$
  - $b' = a - z.b \mod q$
- Store

Instruction Modeling

- Prediction : 10335 $\rightarrow$ **2719 cycles**



PQC in the Superscalar CVA6

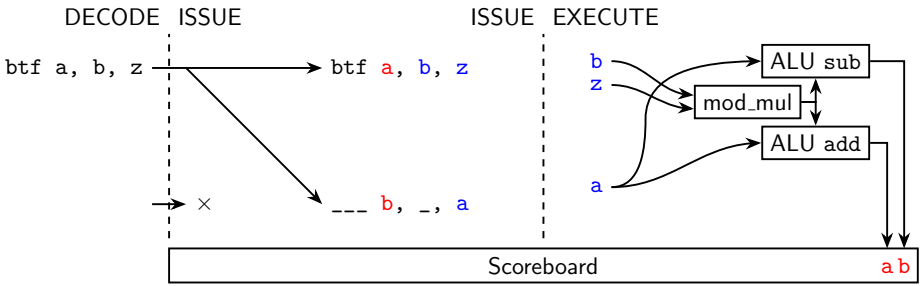
Modeling Phase: Changes in the Model

Operations to perform

- Load
- **Butterfly**
  - $a' = a + z.b \mod q$
  - $b' = a - z.b \mod q$
- Store

Instruction Modeling

- Assembly optimization
- Prediction : 10335 $\rightarrow$ 2719 $\rightarrow$ **2440 cycles**



PQC in the Superscalar CVA6

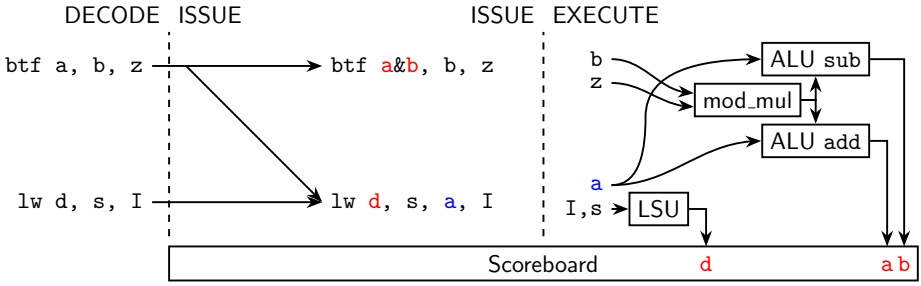
Modeling Phase: Changes in the Model

Operations to perform

- **Load**
- **Butterfly**
  - $a' = a + z.b \mod q$
  - $b' = a - z.b \mod q$
- Store

Instruction Modeling

- Assembly optimization
- Add load instructions in parallel
- Prediction : 10335 $\rightarrow$ 2719 $\rightarrow$ 2440 $\rightarrow$ **2017 cycles**



PQC in the Superscalar CVA6

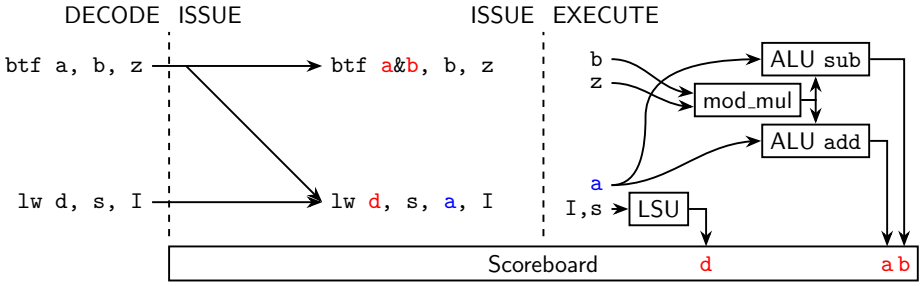
Modeling Phase: Changes in the Model

Operations to perform

- **Load**
- **Butterfly**
  - $a' = a + z.b \mod q$
  - $b' = a - z.b \mod q$
- Store

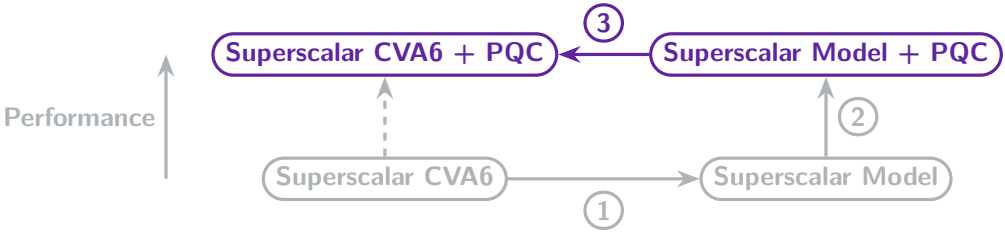
Instruction Modeling

- Assembly optimization
- Add load instructions in parallel
- Add triple-commit
- Prediction : 10335 $\rightarrow$ 2719 $\rightarrow$ 2440 $\rightarrow$ 2017 $\rightarrow$ **1747 cycles**



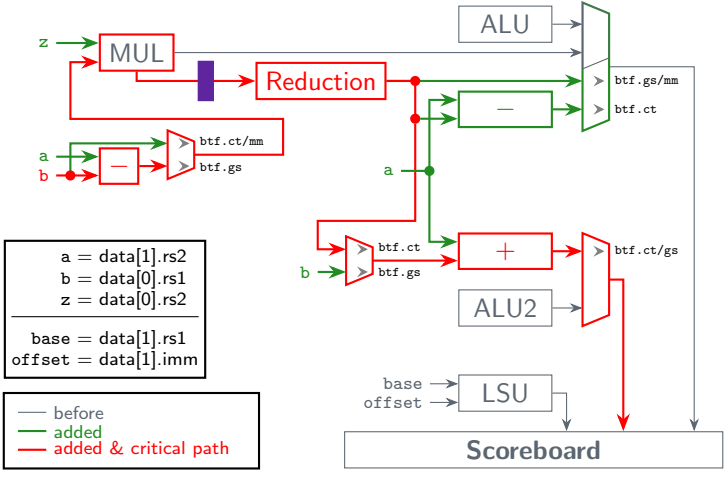
PQC in the Superscalar CVA6

Development Phase



PQC in the Superscalar CVA6

Development Phase



PQC in the Superscalar CVA6

RTL Simulation Results

Duration of NTT, NTT^{-1} and pointwise multiplication (cycles)

Processor	Solution	NTT	NTT^{-1}	Pt.wise Mult.
Superscalar CVA6	32-bit RISC-V	**10335**	**11333**	**2188**
	Our solution	**1913**	**1892**	**807**
	Variation	**-81 %**	**-83 %**	**-63 %**
RI5CY	32-bit RISC-V	17041	20372	4346
	PQ.V.ALU.E	2705	3979	1274
	Variation	-84 %	-80 %	-71 %
Single-issue CVA6	64-bit RISC-V	38043	46266	/
	NTT Instructions	18554	21375	/
	Variation	-51 %	-54 %	/

PQC in the Superscalar CVA6

Hardware Implementation Results

Area

Hardware	Area (cost)	Maximum Frequency Reduction
Superscalar CVA6	109.4 $kGates_{eq}$ (reference)	(reference)
Our solution	114.1 $kGates_{eq}$ (**+4.7 $kGates_{eq}$**)	-2.8 %
RI5CY	43.7 $kGates_{eq}$ (reference)	(reference)
PQ.V.ALU.E	49.8 $kGates_{eq}$ (**+6.8 $kGates_{eq}$**)	Not on critical path

Power

Hardware	Avg. power consumption var.	Energy consumption var.
Superscalar CVA6	(reference)	(reference)
Our solution	**+21 %**	**-78 %**

Conclusion

Conclusion

Goal: **Improve performance of CVA6, especially for PQC**

Model-guided method

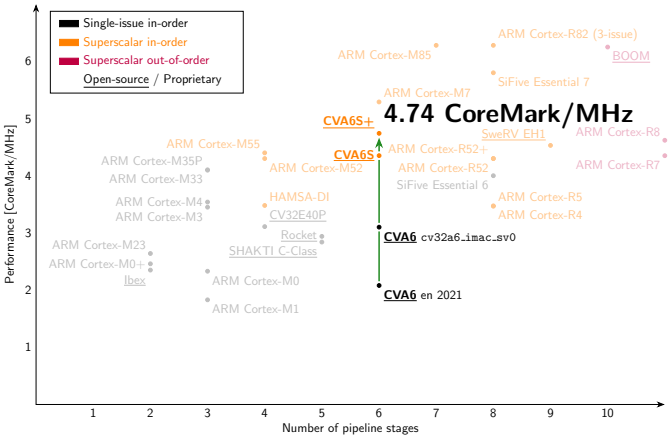
- Performance modeling allows **evaluating solutions before development**
- Then, the model is used as a **reference during development**
- A first method was proposed to model microarchitectural modifications
- This method was extended to add complex instructions to the ISA

Results

- Added the **superscalar** feature to CVA6: **+40% CoreMark/MHz**
- Added **cryptographic instructions**: **NTT runs 5 times as fast**

Conclusion

Contributions to the CVA6 Project

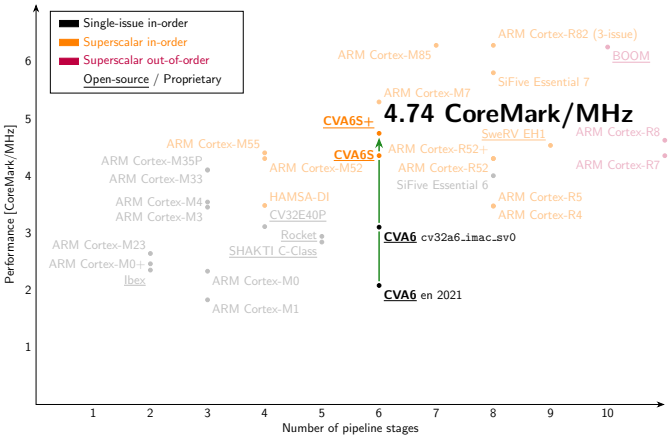


95 accepted contributions, including

- Superscalar
- WAW elimination
- Performance model

Conclusion

Contributions to the CVA6 Project



95 accepted contributions, including

- Superscalar
- WAW elimination
- Performance model
- Configurability improvements
- Documentation
- Automation (scripts and CI)

Conclusion

Dissemination

Publications in conferences

- *"Using a Performance Model to Implement a Superscalar CVA6"*. 21st ACM International Conference on Computing Frontiers. 2024
- *"CVA6S+: A Superscalar RISC-V Core with High-Throughput Memory Architecture"*. RISC-V Summit Europe. 2025. En collaboration avec l'université de Bologne
- *"Eliminating Write-After-Write Hazards to Improve Performance in Embedded Processors"*. 28th Euromicro Conference on Digital System Design (DSD). 2025
- *"Leveraging a Superscalar CVA6 to implement NTT instructions for Post-Quantum Cryptography"*. Constructive Approaches for SeCurity Analysis and Design of Embedded (CASCADE). 2026

Conclusion

Dissemination

Publications in conferences

- *"Using a Performance Model to Implement a Superscalar CVA6"*. 21st ACM International Conference on Computing Frontiers. 2024
- *"CVA6S+: A Superscalar RISC-V Core with High-Throughput Memory Architecture"*. RISC-V Summit Europe. 2025. En collaboration avec l'université de Bologne
- *"Eliminating Write-After-Write Hazards to Improve Performance in Embedded Processors"*. 28th Euromicro Conference on Digital System Design (DSD). 2025
- *"Leveraging a Superscalar CVA6 to implement NTT instructions for Post-Quantum Cryptography"*. Constructive Approaches for SeCurity Analysis and Design of Embedded (CASCADE). 2026

Posters

- *"Performance Modeling of CVA6 with Cycle-Based Simulation"*. RISC-V Summit Europe. 2023
- *"Using a Performance Model to Implement a Superscalar CVA6"*. RISC-V Summit North America. 2025

CVA6 performance

- Adapting the compiler to the superscalar CVA6: currently **4.90 CoreMark/MHz** (outperformed by GCC 15: **5.01 CoreMark/MHz**)
- Method to automatically detect opportunities to create new instructions

Conclusion

Perspectives

CVA6 performance

- Adapting the compiler to the superscalar CVA6: currently **4.90 CoreMark/MHz** (outperformed by GCC 15: **5.01 CoreMark/MHz**)
- Method to automatically detect opportunities to create new instructions

Post-Quantum Cryptography

- Add instructions for ML-KEM, with the appropriate q value
- Perform side-channel and fault injection evaluations of CVA6
- Secure cryptographic instructions against hardware attacks

Conclusion

Perspectives

CVA6 performance

- Adapting the compiler to the superscalar CVA6: currently **4.90 CoreMark/MHz** (outperformed by GCC 15: **5.01 CoreMark/MHz**)
- Method to automatically detect opportunities to create new instructions

Post-Quantum Cryptography

- Add instructions for ML-KEM, with the appropriate q value
- Perform side-channel and fault injection evaluations of CVA6
- Secure cryptographic instructions against hardware attacks

Model

- Using the model for verification
- Creating a model of CVA6 power consumption

Questions?

https://github.com/openhwgroup/cva6

These activities are supported by the TRISTAN project funded by the Key Digital Technologies Joint Undertaking (KDT JU) under grant agreements 101095947. The present action reflects only the authors' view; the European Commission and the JU are not responsible for any use that may be made of the information it contains.

These activities are supported by the ISOLDE project funded by the Key Digital Technologies Joint Undertaking (KDT JU) under grant agreements 101112274. The present action reflects only the authors' view; the European Commission and the JU are not responsible for any use that may be made of the information it contains.