



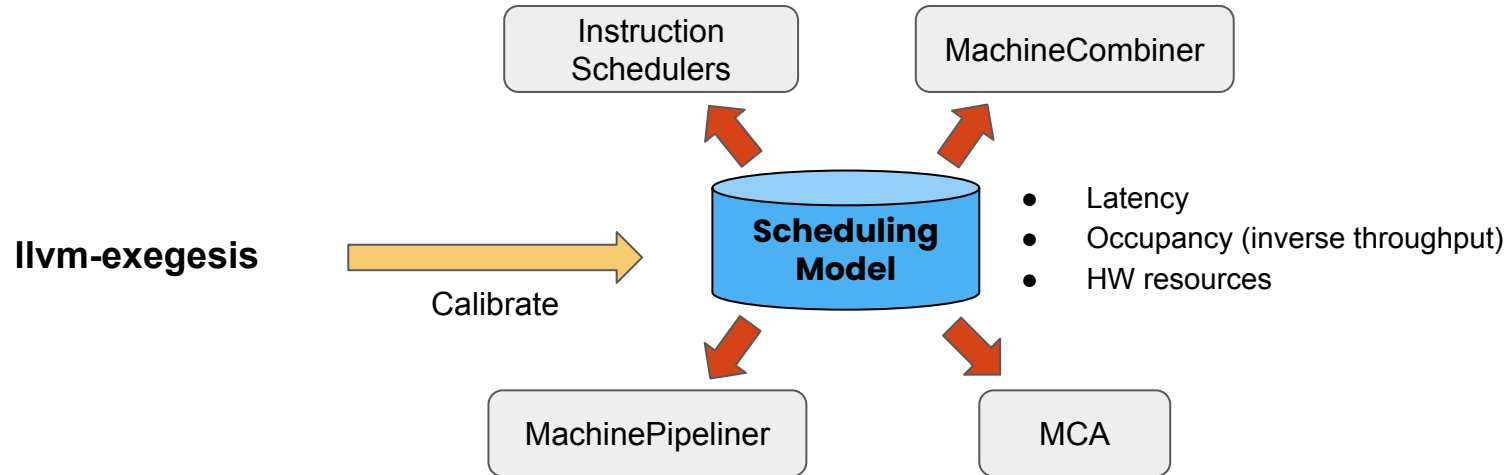
# New llvm-exegesis Support for RISC-V Vector Extension

By Min Hsu

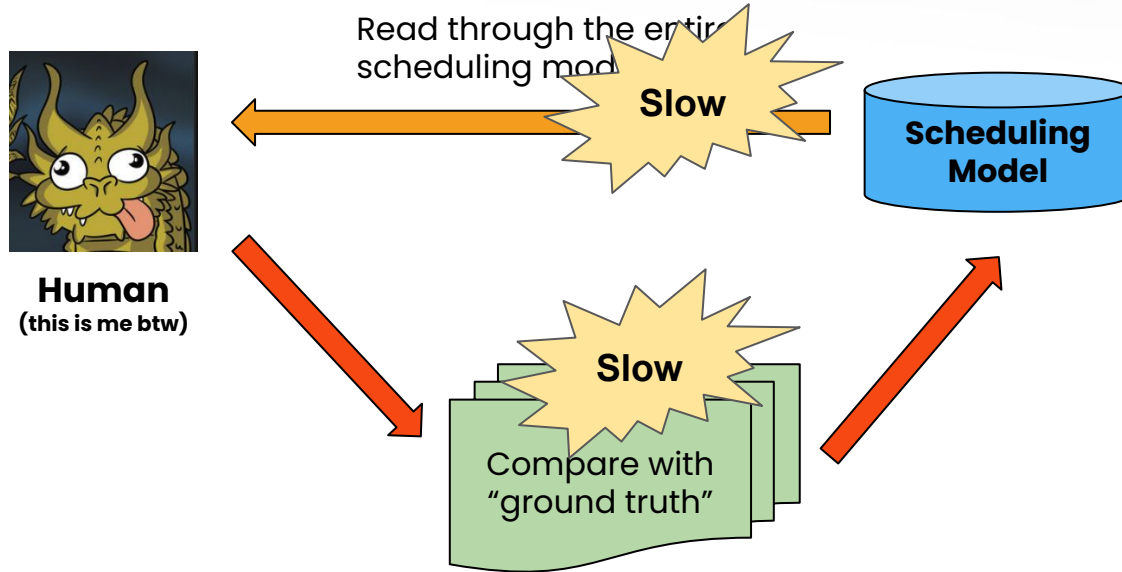
# Highlights

- ◆ The importance of llvm-exegesis in performance modeling for RISC-V
- ◆ The challenges of adding **RISC-V Vector (RVV)** support into llvm-exegesis and our solutions to them
- ◆ Scaling up llvm-exegesis's overall efficiencies, especially in *pre-silicon* RISC-V developments

# llvm-exegesis and Scheduling Model

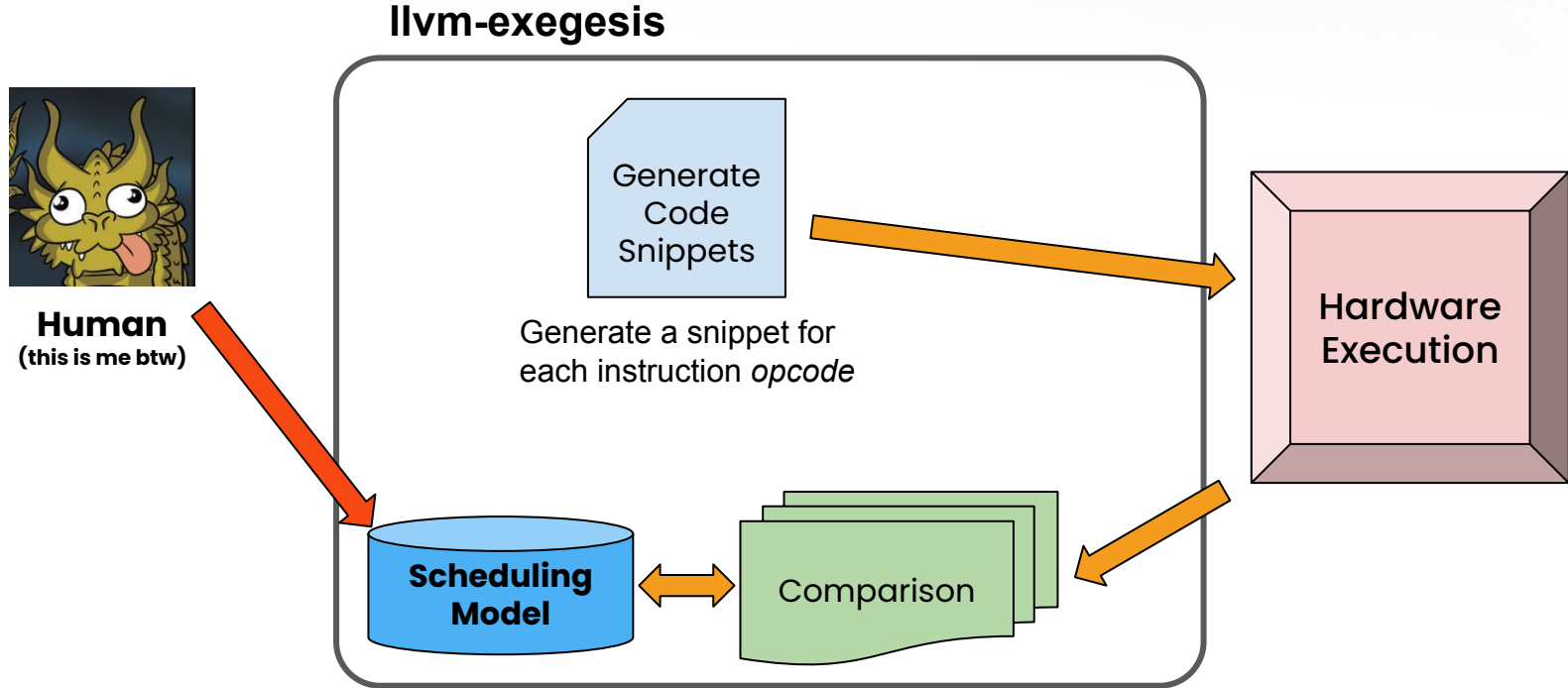


# Before: Calibrate Scheduling Models *Manually*



- Publicly available documents (e.g. AMD SOG)
- Hand-written microbenchmarks (e.g. Agner)
- Consult HW folks (if possible)

# New: Calibrate Scheduling Models *Automatically*



# Benchmarking with llvm-exegesis: an example

Generated snippet: latency

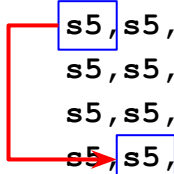
```

---
mode:          latency
key:
  instructions:
    - 'ADD X21 X10 X21'
  config:      ''
  register_initial_values:
    - 'X10=0x0'
    - 'X21=0x0'
cpu_name:     sifive-p670
llvm_triple:  riscv64
min_instructions: 10000
measurements: []
error:       actual measurements skipped.
...
---
```

Snippet for measurement

```

li      a0,0
li      s5,0
add     s5,s5,a0
add     s5,s5,a0
add     s5,s5,a0
add     s5,s5,a0
add     s5,s5,a0
add     s5,s5,a0
... (repeat 10000 times)
```



# Benchmarking with llvm-exegesis: an example

Generated snippet: inverse throughput

```
---
mode:          inverse_throughput
key:
  instructions:
    - 'ADD X8 X19 X23'
  config:      ''
  register_initial_values:
    - 'X19=0x0'
    - 'X23=0x0'
cpu_name:      sifive-p670
llvm_triple:   riscv64
min_instructions: 10000
measurements:  []
error:        actual measurements skipped.
...
---
```

Snippet for measurement

```
li      s3,0
li      s7,0
add     s0,s3,s7
add     s0,s3,s7
add     s0,s3,s7
add     s0,s3,s7
add     s0,s3,s7
add     s0,s3,s7
... (repeat 10000 times)
```

# Benchmarking with llvm-exegesis: an example

Reporting inconsistencies

## llvm-exegesis Analysis Results

**Triple:** riscv64-unknown-linux-gnu

**Cpu:** sifive-p450

**Epsilon:** 0.90

Sched Class `writeCPOP_readCPOP` contains instructions whose performance characteristics do not match that of LLVM:

ClusterId	Opcode/Config	latency
1	CPOP	2.08 [2.08;2.08]

Measured



llvm SchedModel data:

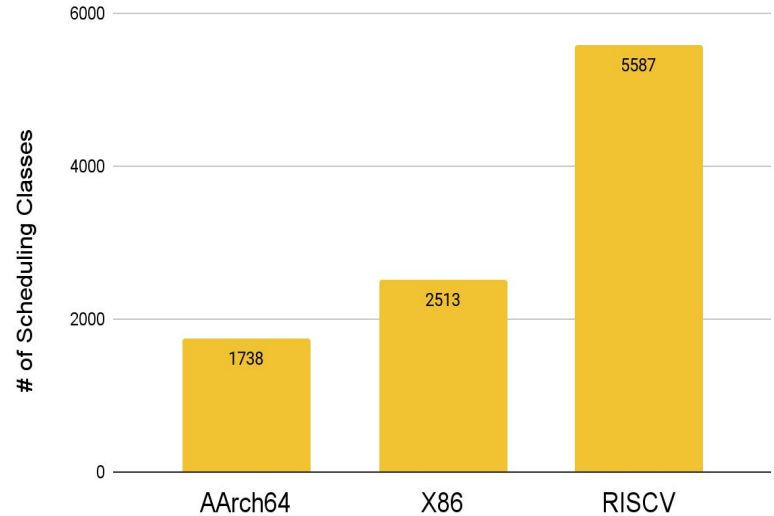
Current scheduling data

Valid	Variant	NumMicroOps	Normalized Latency	RThroughput	WriteProcRes	Idealized Resource Pressure
✓	×	1	3	1.00	SiFiveP400IEXQ2: [0, 1]	SiFiveP400IEXQ2: 1.00



# RISC-V & llvm-exegesis

- ◆ Scheduling class is the smallest unit in a scheduling model. RISC-V has **2.2x** more scheduling classes than X86
- ◆ Majority of these classes are designated to *RISC-V Vector Extension (RVV)* instructions
  - ◇ RVV instructions are of paramount importance to performance
- ◆ Folks from SyntaCore have tried to upstream RISC-V llvm-exegesis support for *scalar* instructions ([#89047](#))
- ◆ **We're presenting our support for RVV instructions in llvm-exegesis**



# Scheduling properties of RVV

## RVV Assembly

```
vadd    v0, v0, v4
```

```
vadd    v0, v0, v4
```

- Each RVV instruction's **configurations (VTYPE)** can be updated *dynamically* during runtime
  - E.g. Element type (SEW), register grouping (LMUL), and number of vector elements (VL)

# Scheduling properties of RVV

## RVV Assembly

```
vsetvli a0, a7, e32, m2, tu, mu
vadd    v0, v0, v4
```

```
vsetvli a0, a7, e64, m4, tu, mu
vadd    v0, v0, v4
```

- Each RVV instruction's **configurations (VTYPE)** can be updated *dynamically* during runtime
  - E.g. Element type (SEW), register grouping (LMUL), and number of vector elements (VL)

# Scheduling properties of RVV

## RVV Assembly

```
vsetvli a0, a7, e32, m2, tu, mu
vadd    v0, v0, v4
```

```
vsetvli a0, a7, e64, m4, tu, mu
vadd    v0, v0, v4
```

- Each RVV instruction's **configurations (VTYPE)** can be updated *dynamically* during runtime
  - E.g. Element type (SEW), register grouping (LMUL), and number of vector elements (VL)
- The same instruction might have completely different latency or inverse throughput under different VTYPE

# Scheduling properties of RVV

## RVV Assembly

```
vsetvli a0, a7, e32, m2, tu, mu
vadd    v0, v0, v4
```

```
vsetvli a0, a7, e64, m4, tu, mu
vadd    v0, v0, v4
```

- **Pseudo instructions** with VTYPE (e.g. SEW, VL) as explicit *operands*

## Machine IR

```
$v0m2 = PseudoVADD_VV_M2 undef $v0m2, $v0m2, $v4m2, $x10 /* v1 */, 5 /* e32 */, 0 /* tu, mu */
```

```
$v0m4 = PseudoVADD_VV_M4 undef $v0m4, $v0m4, $v4m4, $x10 /* v1 */, 6 /* e64 */, 0 /* tu, mu */
```

# Scheduling properties of RVV

## RVV Assembly

```
vsetvli a0, a7, e32, m2, tu, mu
vadd    v0, v0, v4
```

```
vsetvli a0, a7, e64, m4, tu, mu
vadd    v0, v0, v4
```

- **Pseudo instructions** with VTYPE (e.g. SEW, VL) as explicit *operands*
- A LMUL-based pseudo instruction & scheduling class design
  - Most RVV instructions have varying scheduling properties in different LMULs

## Machine IR

```
$v0m2 = PseudoVADD_VV_M2 undef $v0m2, $v0m2, $v4m2, $x10 /* v1 */, 5 /* e32 */, 0 /* tu, mu */
```

```
$v0m4 = PseudoVADD_VV_M4 undef $v0m4, $v0m4, $v4m4, $x10 /* v1 */, 6 /* e64 */, 0 /* tu, mu */
```

# RVV support in llvm-exegesis

High-level design

- ◆ A custom snippet generator that enumerates every single **RVV pseudo** opcodes
  - ◇ For each opcode, enumerate all possible (*legal*) SEW, VL, FRM / VXRm, and tail/mask policies via the pseudo instruction operands

# RVV support in llvm-exegesis

## High-level design

- ◆ A custom snippet generator that enumerates every single **RVV pseudo** opcodes
  - ◇ For each opcode, enumerate all possible (*legal*) SEW, VL, FRM / VXRM, and tail/mask policies via the pseudo instruction operands
- ◆ Run MachineIR Passes on the generated snippets
  - ◇ RISCVInsertVSETVLIPass – insert vsetvli instructions that match the VTYPE
  - ◇ RISCVInsertWriteVXRMPass – insert VXRM update instructions
  - ◇ Custom post-processing Pass – an Exegesis-specific Pass to cleanup some remaining virtual registers

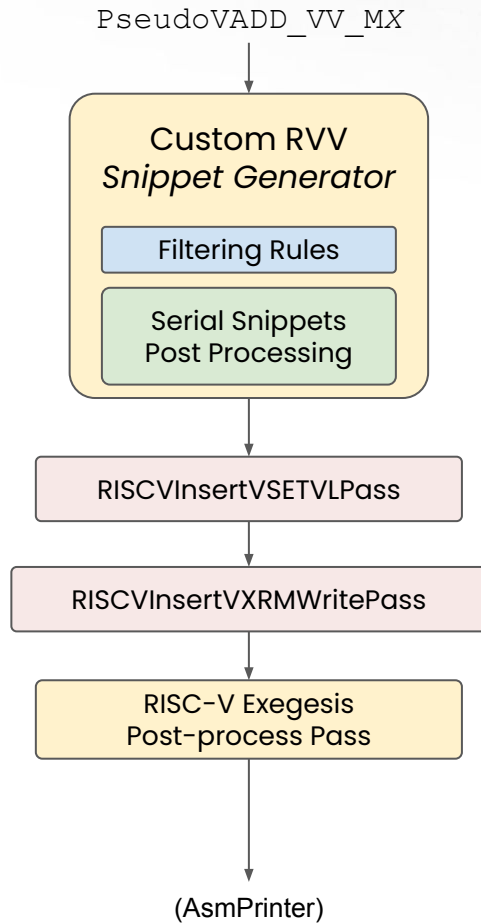


# Generate legal RVV snippets

- ◆ Filtering out illegal VTYPE combinations
  - ◆ Fractional LMULs (e.g. MF2) only support some of the SEW
  - ◆ Instructions that disallow overlapping source / dest register group
  - ◆ Vector crypto (Zvk\*) specific rules (e.g. the EGW constraint)

# Generate legal RVV snippets

- ◆ Filtering out illegal VTYPE combinations
  - ◇ Fractional LMULs (e.g. MF2) only support some of the SEW
  - ◇ Instructions that disallow overlapping source / dest register group
  - ◇ Vector crypto (Zvk\*) specific rules (e.g. the EGW constraint)
- ◆ The *passthru* operand in RVV instructions
  - ◇ Most RVV instructions have an additional passthru operand that Exegesis's serial snippet generator confuses as an input operand (it's not)



← Allow pseudo opcodes

- Enumerate all possible VTYPE combinations
- Filter out ineligible opcodes

← Assign self-aliasing registers for serial snippets

 New component

 Existing Pass

← *Manually* allocate registers for some VSETVL and WriteFRM instructions

← Lower RVV pseudo instructions

# Case study: RVV integer slide up / down

Before

Instructions	Old Scheduling class (relevant part)
<code>vslideup.vx</code> <code>vslidedown.vx</code>	WriteVISlideX_M2/M4/M8

	Old SiFive P470 Scheduling Model	
Instructions	Latency (LMUL = 4)	Inverse Throughput (LMUL = 4)
<code>vslideup.vx</code>	8 cycles	4 cycles
<code>vslidedown.vx</code>		

# Case study: RVV integer slide up / down

Before

Instructions	Old Scheduling class (relevant part)
<code>vslideup.vx</code> <code>vslidedown.vx</code>	WriteVISlideX_M2/M4/M8

Instructions	Old SiFive P470 Scheduling Model		llvm-exegesis Measurements	
	Latency (LMUL = 4)	Inverse Throughput (LMUL = 4)	Latency (LMUL = 4)	Inverse Throughput (LMUL = 4)
<code>vslideup.vx</code>	8 cycles	4 cycles	~12 cycles	
<code>vslidedown.vx</code>			~14 cycles	

# Case study: RVV integer slide up / down

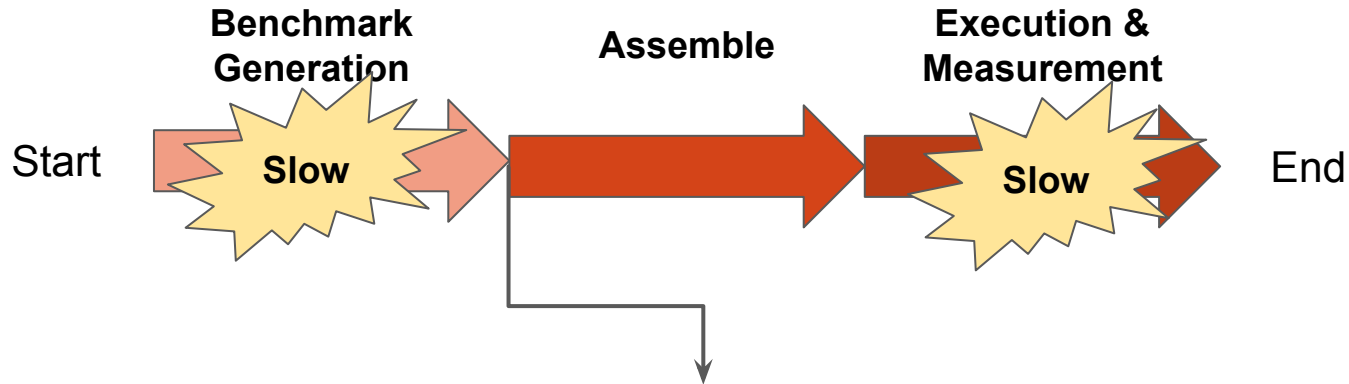
Solution: split the scheduling class

Instructions	Old Scheduling class (relevant part)	New scheduling classes (relevant part)
<code>vslideup.vx</code> <code>vslidedown.vx</code>	WriteVISlideX_M2/M4/M8	WriteVSlideUpX_M2/M4/M8
		WriteVSlideDownX_M2/M4/M8

Instructions	Old SiFive P470 Scheduling Model		Illum-exegesis Measurements	
	Latency (LMUL = 4)	Inverse Throughput (LMUL = 4)	Latency (LMUL = 4)	Inverse Throughput (LMUL = 4)
<code>vslideup.vx</code>	8 cycles	4 cycles	~12 cycles	
<code>vslidedown.vx</code>			~14 cycles	

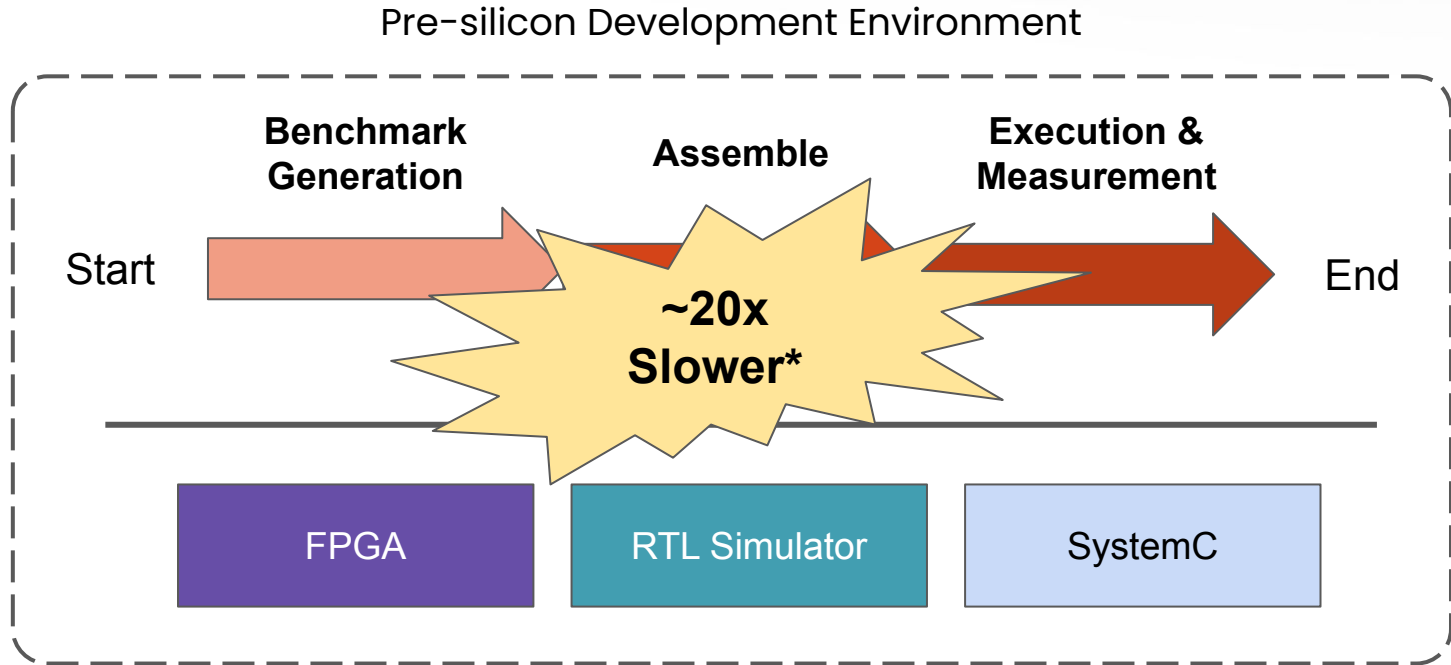
- Patch that splitted the scheduling class: [7064e4b](#)
- The new latency & inverse throughput info is part the P400 scheduling model update ([7efa068](#))

# Challenge: llvm-exegesis's monolithic workflow



- **32K** different RVV snippets for latency
- **131K** different RVV snippets for inverse throughput

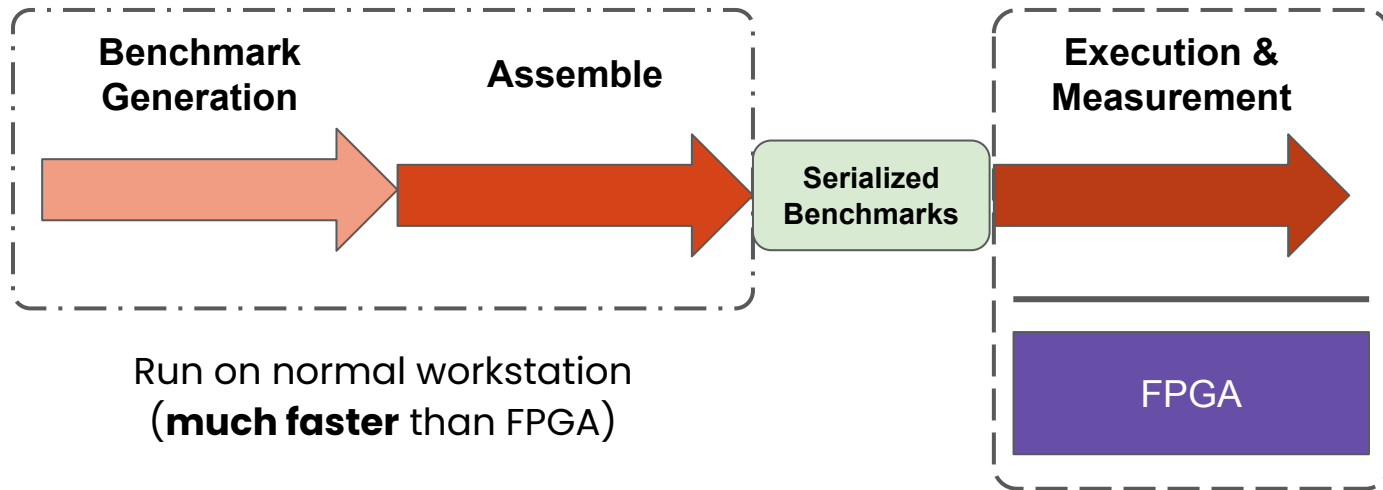
# Challenge: llvm-exegesis's monolithic workflow



\*: Compare between the time measured inside FPGA & the actual wall-clock time in the outside world

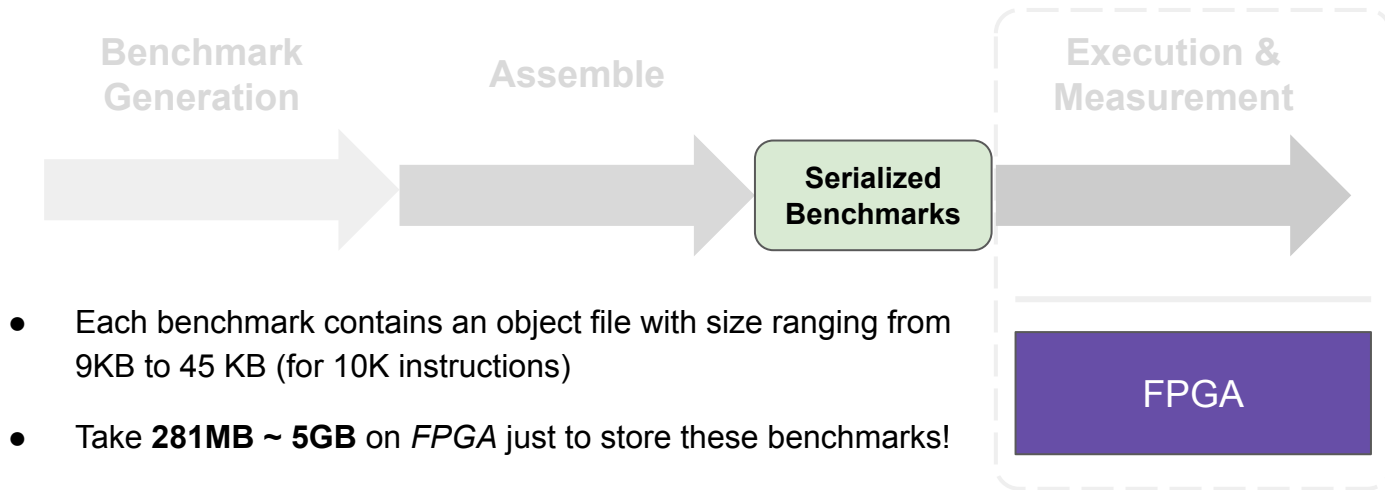


# Solution: Generate snippets ahead of time



# Solution: Generate snippets ahead of time

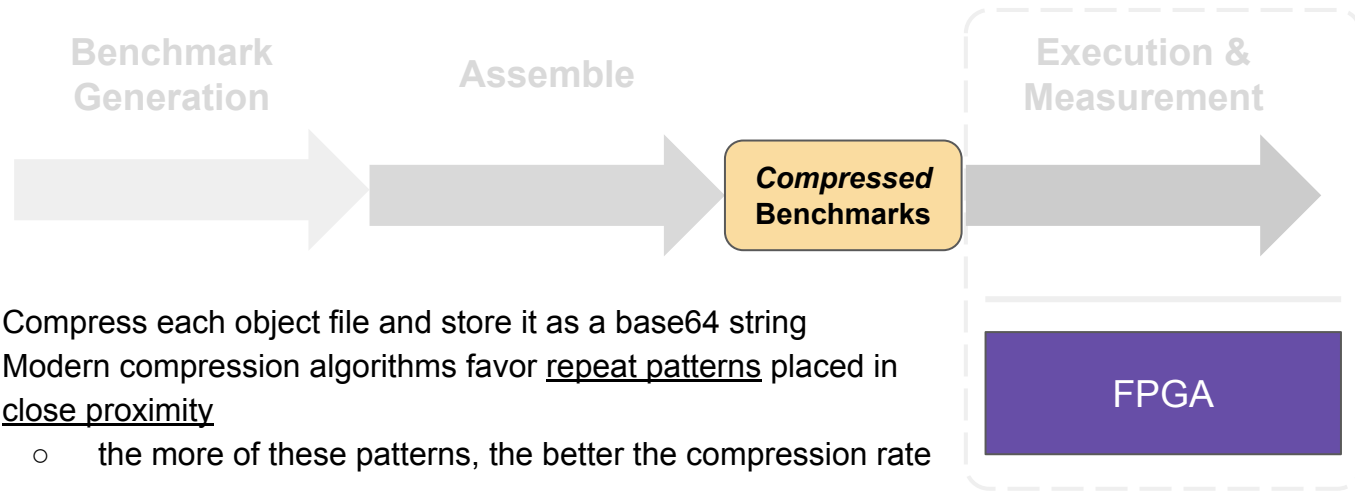
Naive approach: dump all the benchmark object files



- Each benchmark contains an object file with size ranging from 9KB to 45 KB (for 10K instructions)
- Take **281MB ~ 5GB** on *FPGA* just to store these benchmarks!
  - Doesn't scale well if we want to increase the # of instructions or # of benchmarks

# Solution: Generate snippets ahead of time

Compression to the rescue!

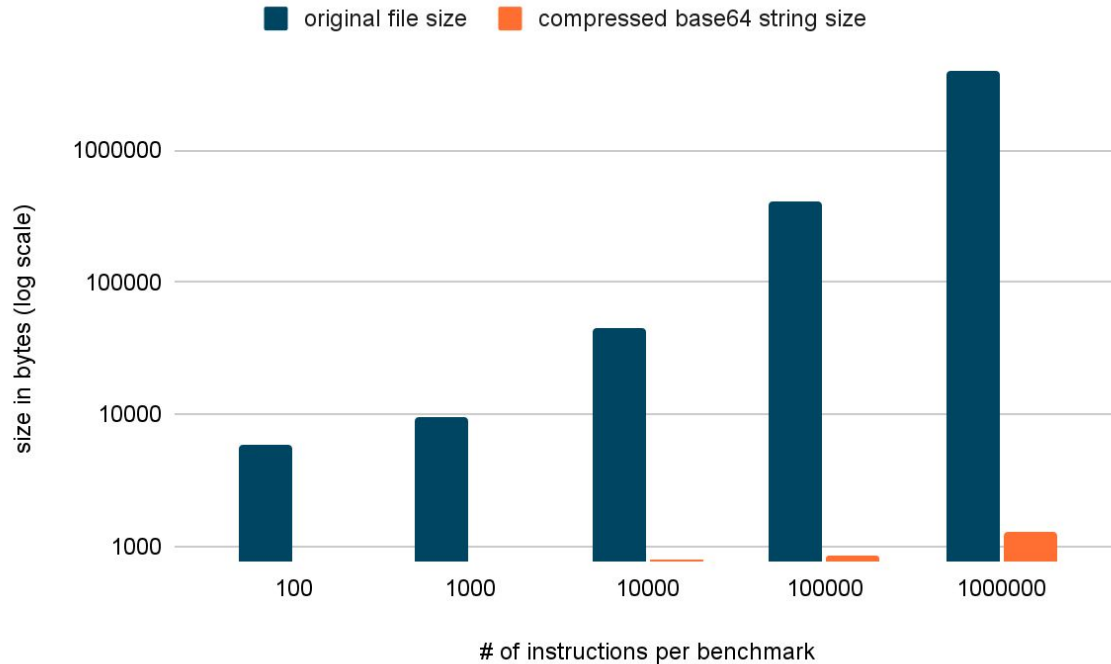


- Compress each object file and store it as a base64 string
- Modern compression algorithms favor repeat patterns placed in close proximity
  - the more of these patterns, the better the compression rate

## Key Insight

Each benchmark has 10000 *(nearly) identical* instructions placed *side-by-side*!

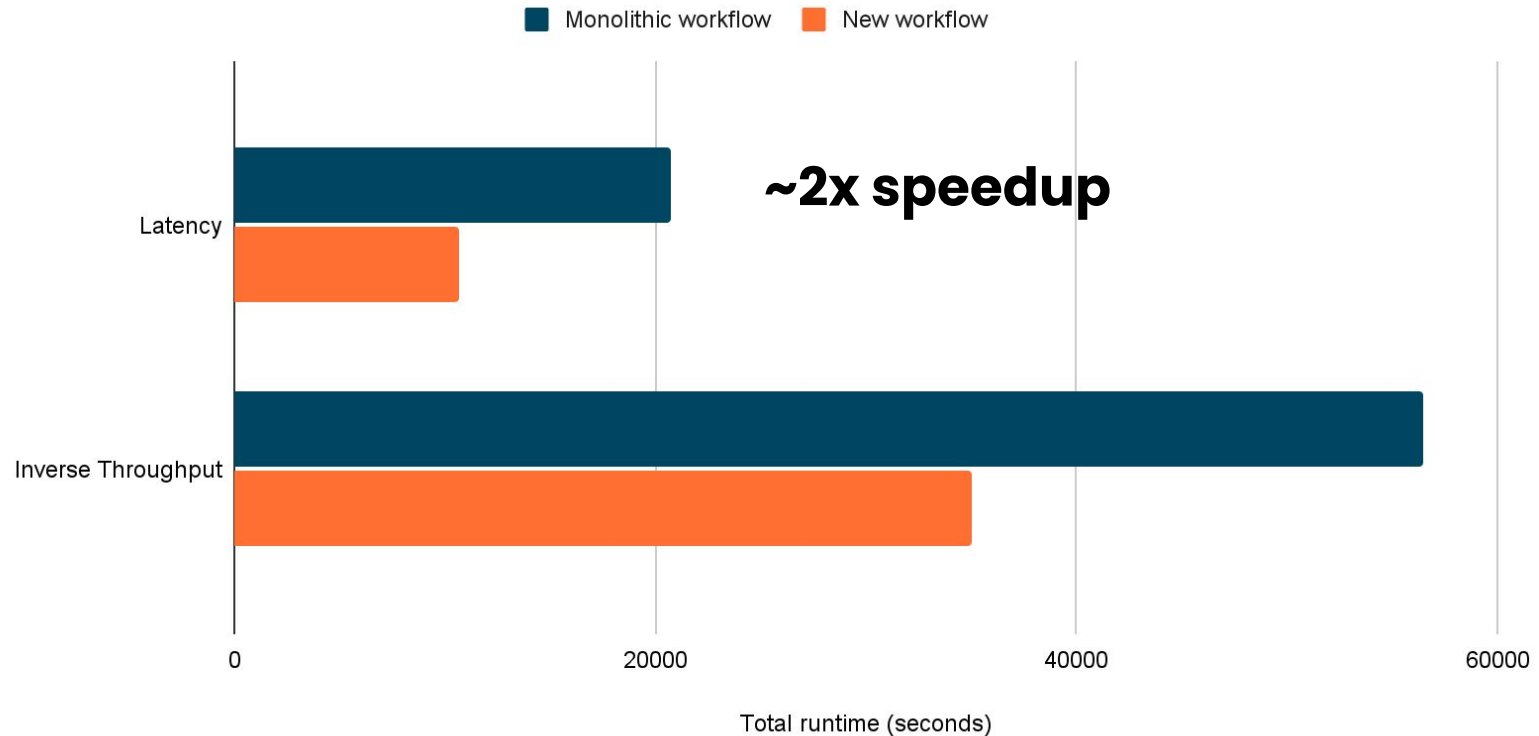
# Improved compression efficiency



# of insts per benchmark	Space saving rate
100	87.09%
1000	91.96%
10000	<b>98.27%</b>
100000	<b>99.79%</b>
1000000	<b>99.97%</b>

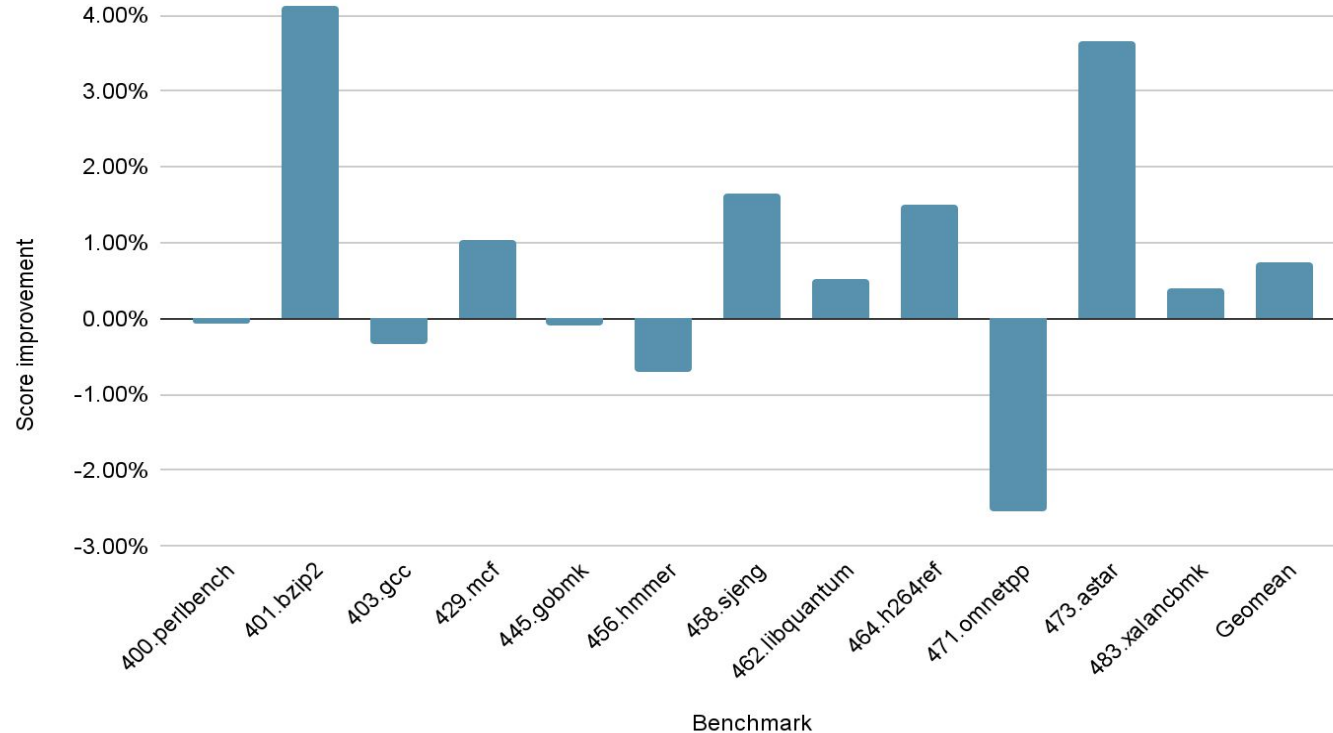
# Improved llvm-exegesis efficiency

Generating snippets for SiFive P670



# Scheduling model improvement

SPEC2006 INT on SiFive P470



# Limitations & Future Plans

- ◆ RVV memory instructions are currently not supported
- ◆ Some instructions' latency / rthroughput depend on input values (e.g. `vrgather .vv`, divisions) and we're not generating the correct values
- ◆ llvm-exegesis requires Linux, which might be a problem for some (embedded) processors / microcontrollers
  - ◇ Support other means of measurement in the future, like cycle-accurate simulators
- ◆ We would like to **upstream our work** on top of SyntaCore's PR. Though the review is moving slow right now

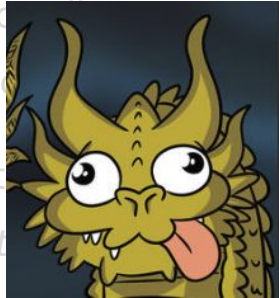
# Summary

- ◆ llvm-exegesis helps us to calibrate the scheduling model for performance-critical RVV instructions *in scale*
- ◆ We created a custom llvm-exegesis pipeline for RVV that delivers good coverages over all possible vector configurations
- ◆ We add a new feature into llvm-exegesis that offloads the snippet generation phase. It gives more *flexibility* to llvm-exegesis and improves its efficiency by about **2x** in our pre-silicon development testbed
- ◆ Our work also improves the scheduling models quality and shows at least **2%** performance improvements in some SPEC2006 benchmarks



# Summary

- ◆ llvm-exegesis helps us to calibrate the scheduling model for performance-critical RVV instructions *in scale*
- ◆ We created a custom analysis pipeline for RVV that delivers good coverages over all possible vector lengths
- ◆ We add a new feature to the key generation phase. It gives more *flexibility* to the analysis and improves its efficiency by **at least 2x** in our pre-silicon development testbed
- ◆ Our work also improves the scheduling models quality and shows at least **2%** performance improvements in some SPEC2006 benchmarks



Thank You!