



Flexible Timing Simulation of RISC-V Processors with Sniper

Neethu Bal Mallya¹, Cecilia Gonzalez-Alvarez², Trevor E. Carlson¹

¹National University of Singapore, Singapore

²Ghent University, Belgium



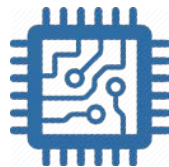
Outline

- Need for Simulation
- Sniper Simulator Overview
- Our enhancements to Sniper
- Initial Processor Performance Analysis
- Conclusion

Why do we need Simulation?



Performance analysis of next-generation systems

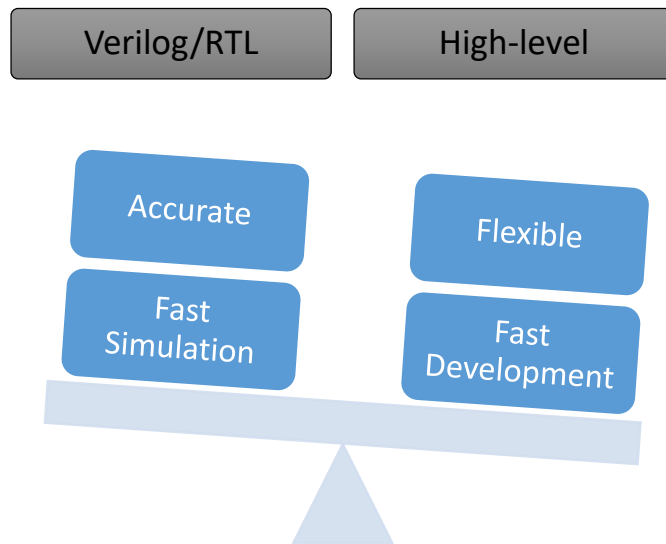


Architecture design space exploration

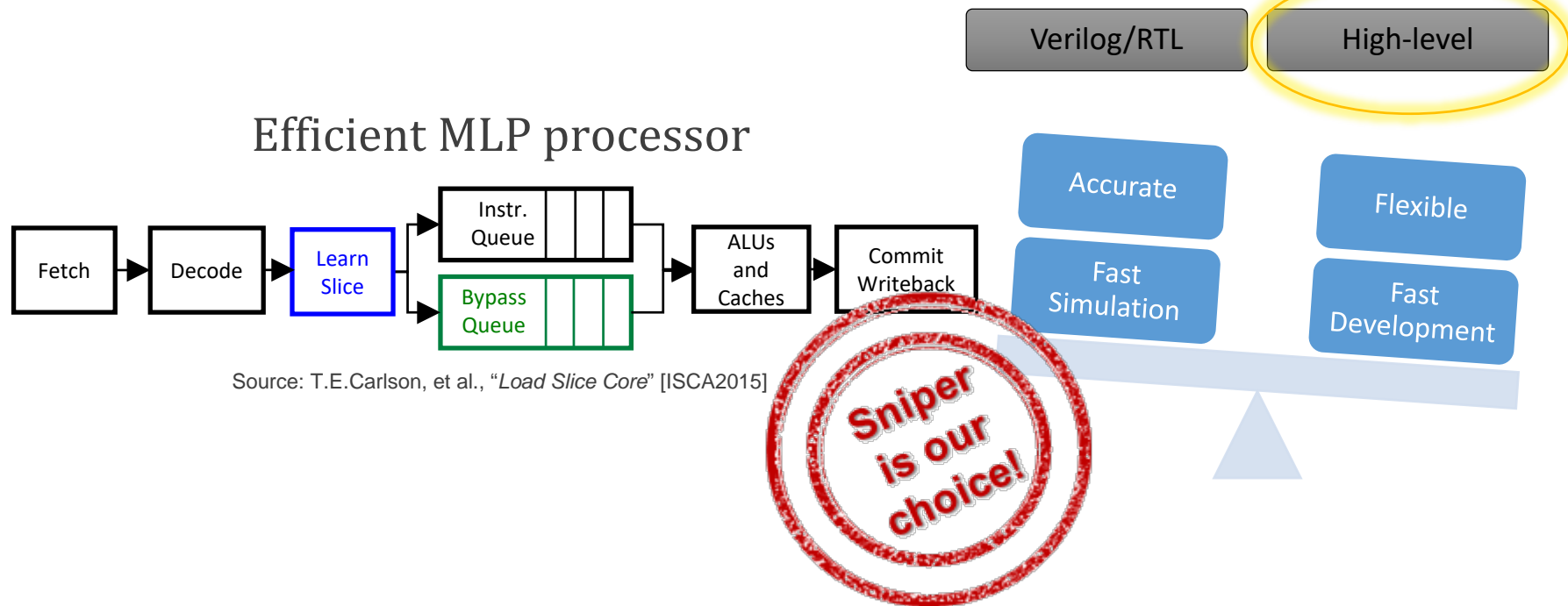


Pre-silicon software optimizations

Trade-offs in Simulation



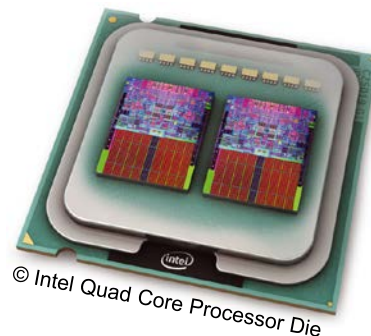
Trade-offs in Simulation



Sniper Simulator – An Overview

- Parallel simulator based on Interval Simulation
- Models multi-/many-cores running multithreaded¹ and multi-program workloads
- Hardware validated for x86
- Flexible simulation options

¹Currently not supported for RISC-V



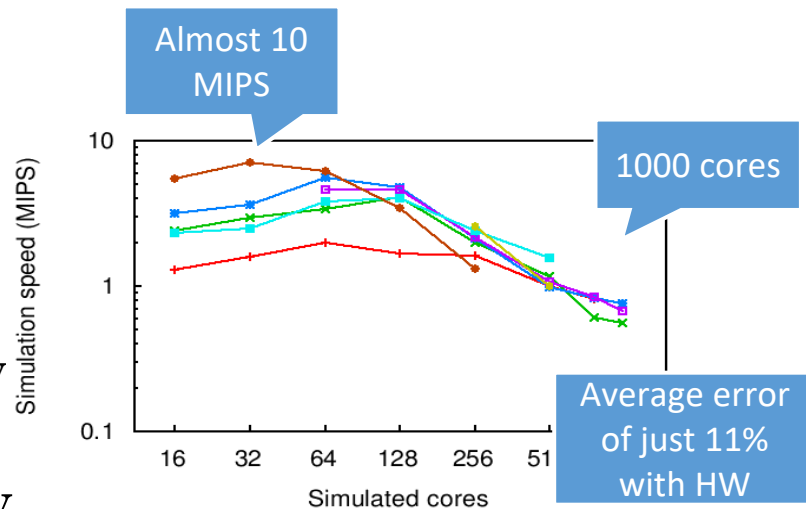
Sniper – Beyond Traditional Simulation

- Strong adoption in industry and academia
 - 550+ citations
 - 800+ researcher downloads
 - 64+ countries
- Actively used since 2011
 - Belgium-based team
 - Supports next generation Xeon Phi (KNL++)
 - HiPEAC TechTransfer Award



Sniper – Key Differentiators

- Fast development time
- Enables Limit Studies
 - Branch Prediction
 - Memory Dependence Prediction
 - Shared Multi-level Cache Hierarchy
- High Performance and Scalability



Sniper - Interacting with the Simulator

- Python interfaces
- SimAPI
 - Magic Instructions
 - SimROIStart() - SimROIEnd()

\$SNIPER_HOME/include/sim_api.h

```
// SimSetInstrumentMode options
#define SIM_OPT_INSTRUMENT_DETAILED 0
#define SIM_OPT_INSTRUMENT_WARMUP 1
#define SIM_OPT_INSTRUMENT_FASTFORWARD 2

// SimAPI commands
SimRoiStart()
SimRoiEnd()
SimGetProcId()
SimGetThreadId()
SimSetThreadName(name)
SimGetNumProcs()
SimGetNumThreads()
SimSetFreqMHz(proc, mhz)
SimSetOwnFreqMHz(mhz)
SimGetFreqMHz(proc)
SimGetOwnFreqMHz()
SimMarker(arg0, arg1)
SimNamedMarker(arg0, str)
SimUser(cmd, arg)
SimSetInstrumentMode(opt)
SimInSimulator()
```

Sniper - Interacting with the Simulator

- Energy Stats

```
class EnergyStats:
    def setup(self, args):
        args = dict(enumerate((args or '').split(':')))
        interval_ns = long(args.get(0, None) or 1000000) # Default power update every 1 ms
        sim.util.Every(interval_ns * sim.util.Time.NS, self.periodic, not_only = True)
        self.dvfs_table = build_dvfs_table(int(sim.config.get('power/technology_node')))
        self.name_last = None
        self.time_last_power = 0
        self.time_last_energy = 0
        self.in_stats_write = False
        self.power = {}
        self.energy = {}
        for metric in ('energy-static', 'energy-dynamic'):
            for core in range(sim.config.ncores):
                sim.stats.register('core', core, metric, self.get_stat)
                sim.stats.register('L1-I', core, metric, self.get_stat)
                sim.stats.register('L1-D', core, metric, self.get_stat)
                sim.stats.register('L2', core, metric, self.get_stat)
                sim.stats.register('processor', 0, metric, self.get_stat)
                sim.stats.register('dram', 0, metric, self.get_stat)
```

```
def periodic(self, time, time_delta):
    self.update()
```

Run
McPAT

```
def update(self):
```

```
    power = self.run_power(self.name_last, current)
    self.update_power(power)
```

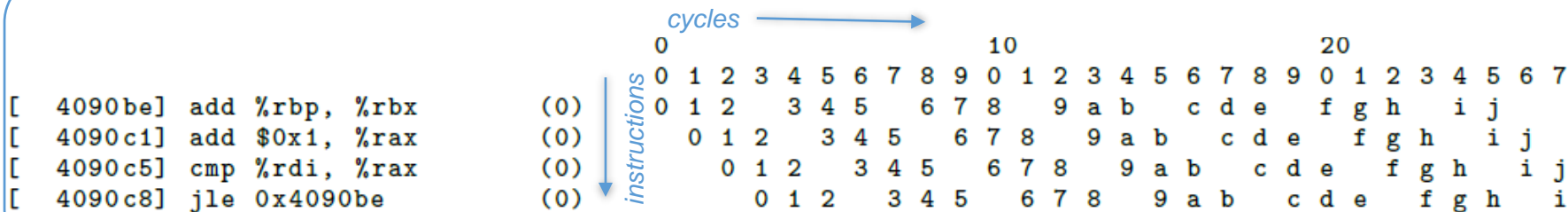
```
    self.update_energy()
```

Update the statistics



Sniper - Interacting with the Simulator

- Loop Tracer

```
[general]
syntax=att                                # Optional
[loop_tracer]
enabled=true
base_address=4090be # Loop start address
iter_start=9000      # Wait before starting
iter_count=20        # Number to view
```



Sniper + RISC-V ecosystem

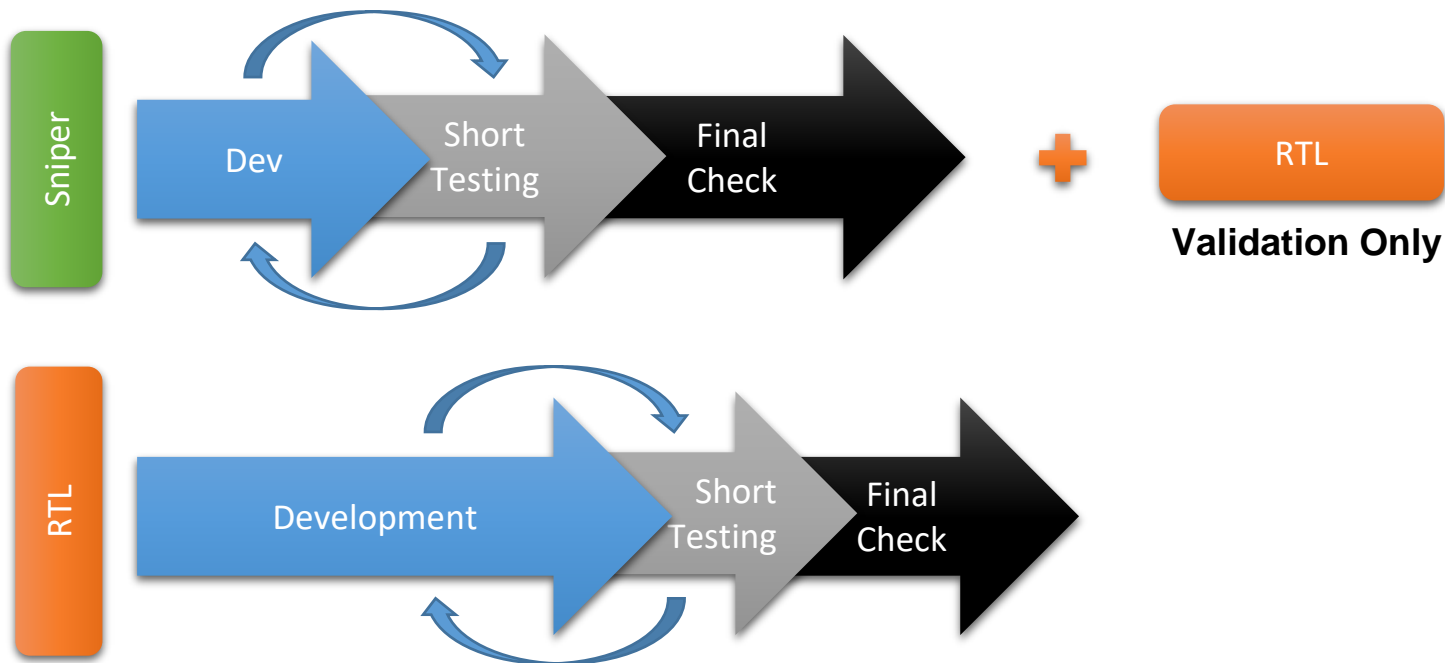
- RISC-V
 - Open, Extensible ISA
 - Collection of related software tools
- Existing Architecture-level Software implementations
 - Functional simulators
 -  Spike
 -  rv8
 - Many additional things



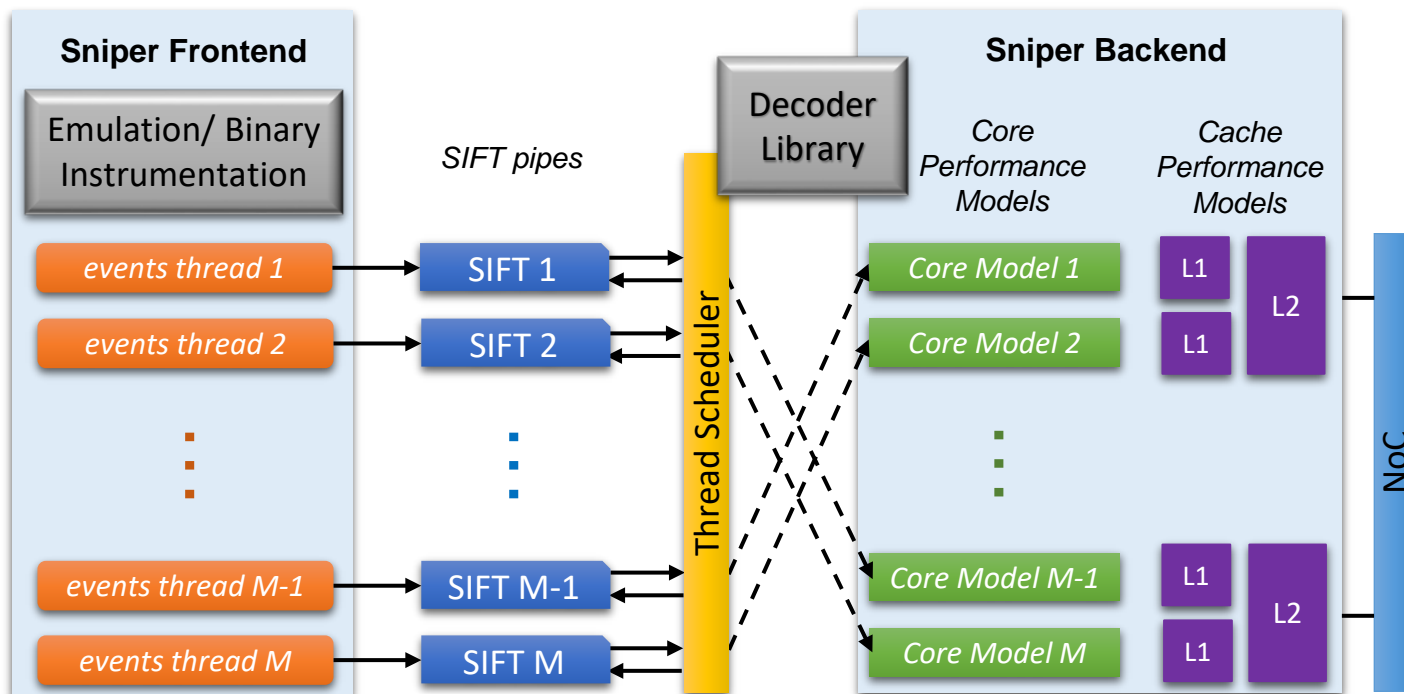
Comparison with existing solutions

	Sniper + RISC-V	gem5 (RISC5)	FireSim / Chisel / Verilog
Development Methodology	C++ based (SW)	C++ based (HW)	RTL based (HW)
Dev-time	+++	++	+
Sim-time	+++	++	++++/+ / +
Simulation model	Cycle-level + Cycle-approximate	Cycle-level	Cycle-exact + Cycle-approximate
Flexibility	Ease-of-use / modification		Requires RTL/ abstract models
Fidelity	Sophisticated models require hardware validation		Cycle-exact models derived from synthesizable RTL

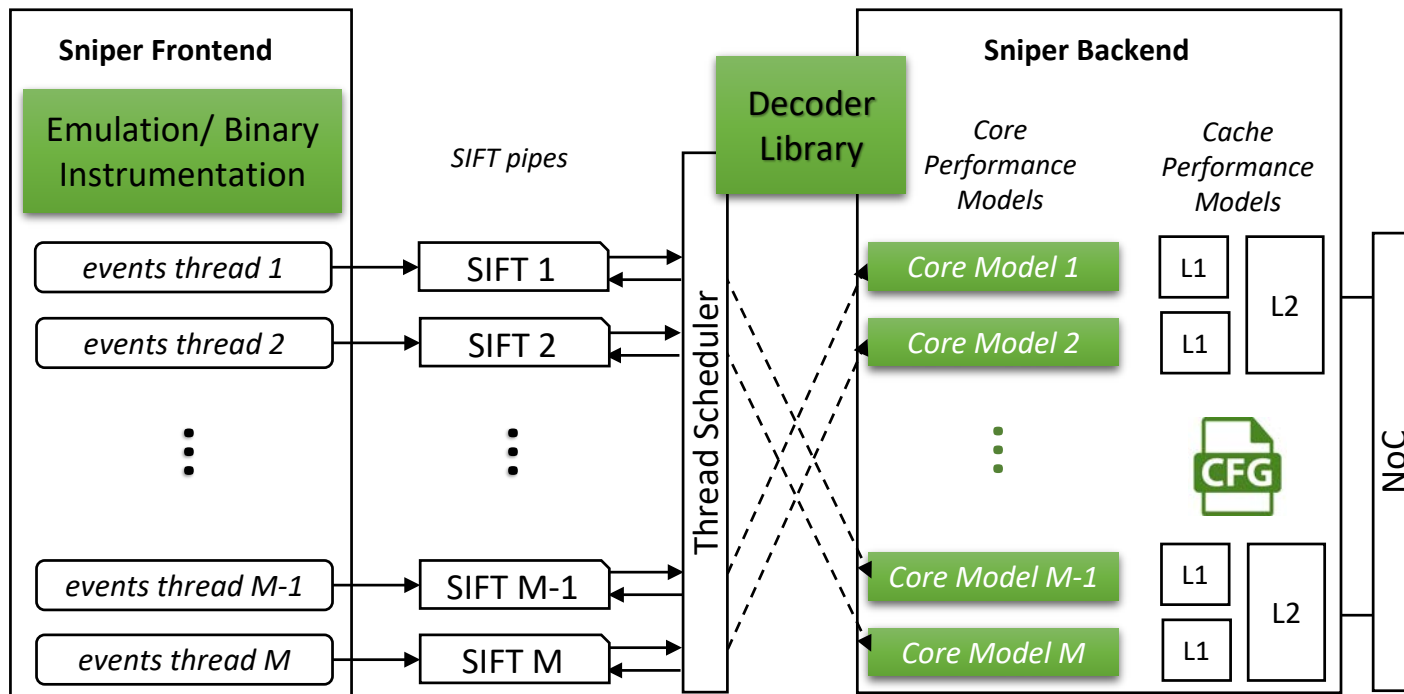
Simulation Flow



Sniper Architecture

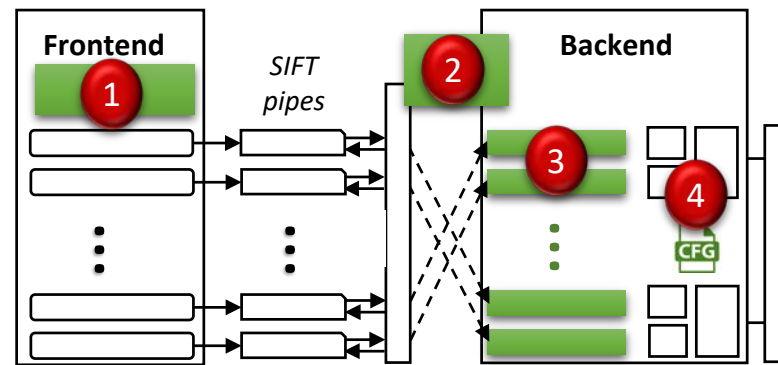


How did we enhance Sniper?



How did we enhance Sniper?

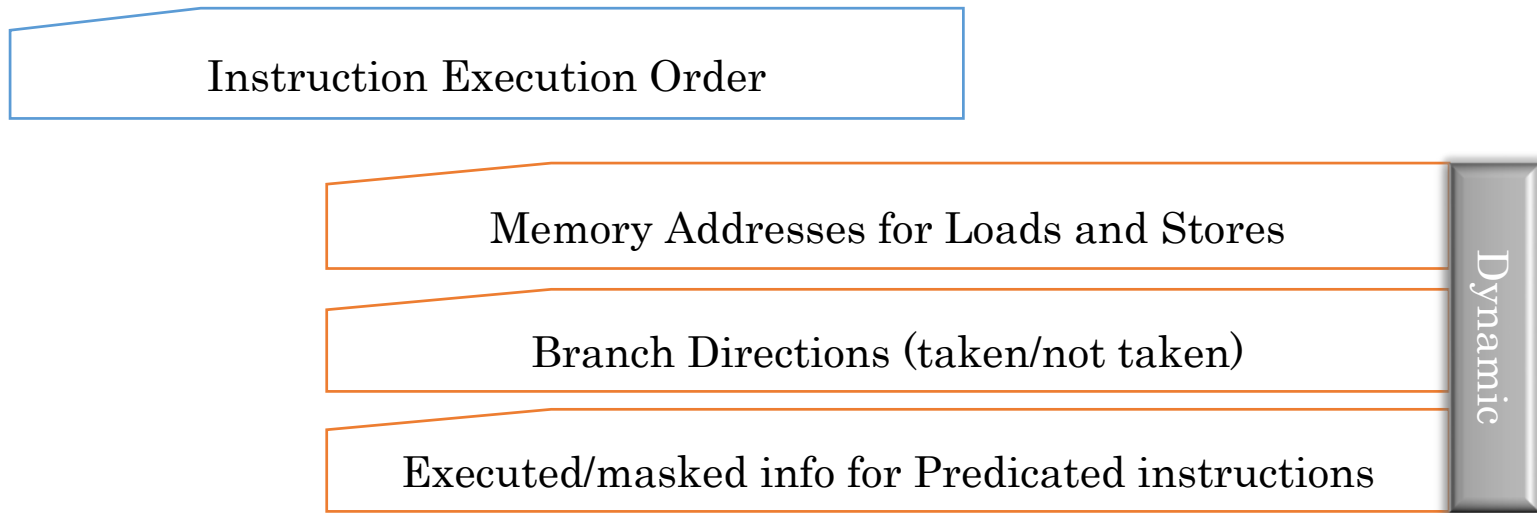
- 1 RISC-V functional simulators - rv8 / Spike were updated to support SIFT generation
- 2 Decoder Library
Architectural agnostic methods were added to implement the decoding phase of the processor
- 3 Core Model
Parameters like description of ports/ functional units, latencies, etc. were updated



- 4 Configuration files
to resemble a BOOM processor

Sniper Instruction Trace File Format (SIFT)

- Dynamic Instruction stream generated by the Frontend



How to add new Frontend?



Sift::
Writer

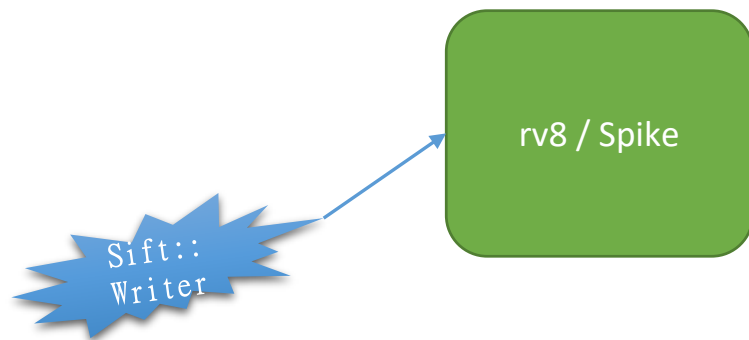
Control

```
Sift::Writer::Magic()
```

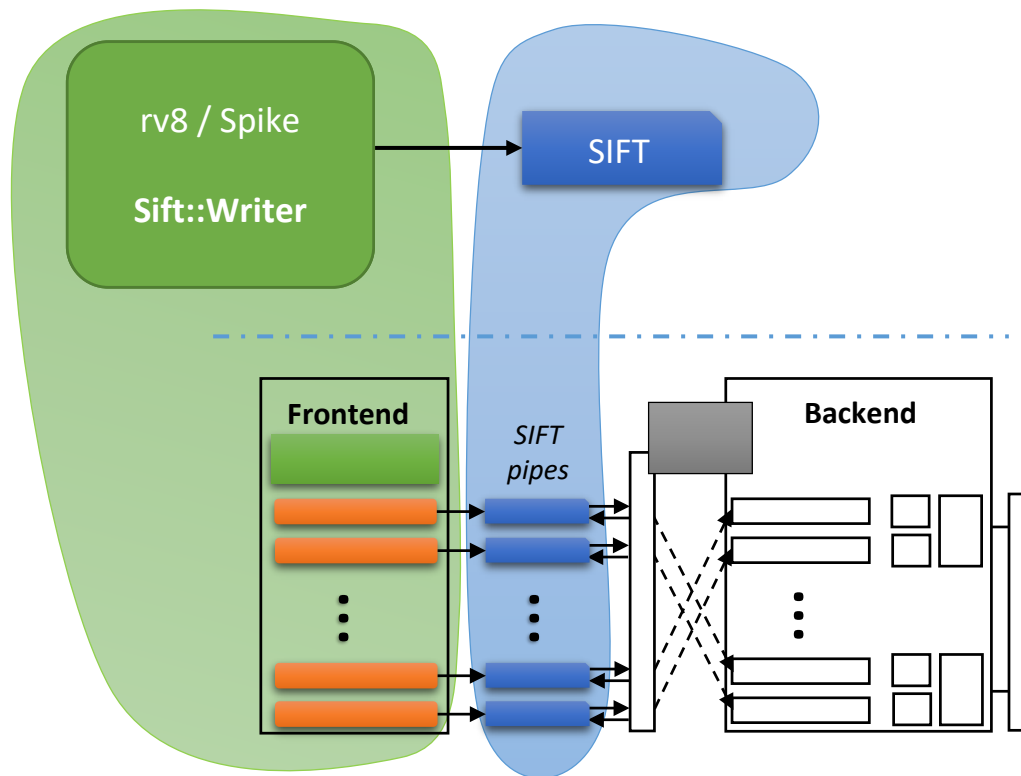
Instruction Instrumentation

```
Sift::Writer::InstructionCount()  
Sift::Writer::CacheOnly()  
Sift::Writer::Instruction()  
    // addresses, branch direction, etc.
```

How to add new Frontend?



How to add new Frontend?



How to update Backend?

- Decoder Library
 - 2 classes
 - Decoder
 - InstructionDecoded

`$SNIPER_HOME/decoder_lib`

- Core Model

`$SNIPER_HOME/common/performance_model`

- Config Files

`$SNIPER_HOME/config`

How to run Sniper ?

`./run-sniper --frontend=[pin | dr | spike | rv8 | legacy] --config`

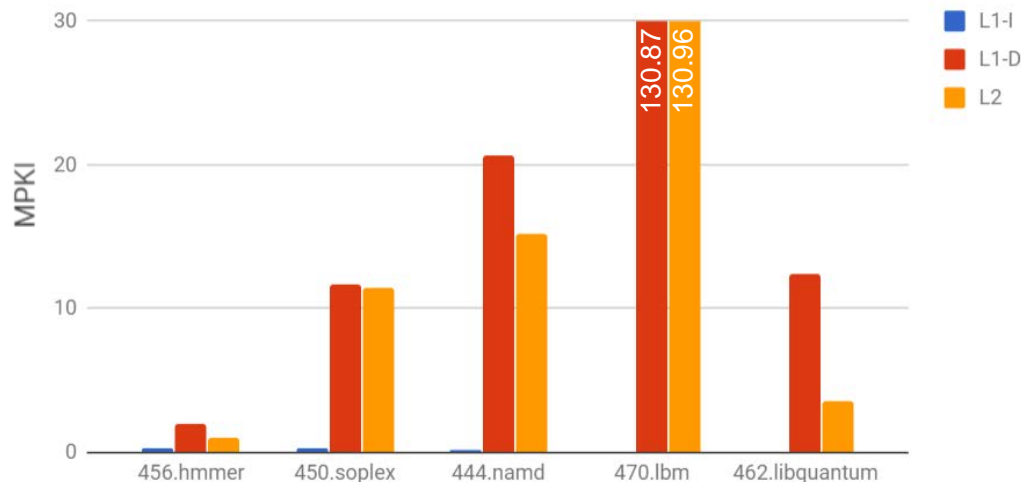
```
[SNIPER] Start
[SNIPER] -----
[SNIPER] Sniper using SIFT/trace-driven frontend
[SNIPER] Running full application in DETAILED mode
[SNIPER] -----
[SNIPER] Enabling performance models
[SNIPER] Setting instrumentation mode to DETAILED
Trace Monitor Started
[TRACE:0] -- DONE --
[SNIPER] Disabling performance models
[SNIPER] Leaving ROI after 18.26 seconds
OUT: RUN: TraceThread
[SNIPER] Simulated 5.0M instructions, 11.2M cycles, 0.45 IPC
[SNIPER] Simulation speed 273.4 KIPS (273.4 KIPS / target core - 3657.1ns/instr)
[SNIPER] Setting instrumentation mode to FAST_FORWARD
[SNIPER] End
[SNIPER] Elapsed time: 18.41 seconds
```

Experimental Setup

- Sniper multi-core simulator
 - Similar to BOOM v1 DefaultConfig
 - Dispatch width:2, Issue Width:3, ROB:80
 - 32KB L1s, 1MB L2
 - 2.0GHz
- SPEC CPU2006 benchmarks
 - First 5M instructions

Initial Processor Performance Analysis

Testcase	IPC	KIPS
470.lbm	0.15	97.899
444.namd	1	304.719
450.soplex	1.52	343.668
456.hmmer	2.71	523.41
462.libquantum	2.65	611.968



Metrics	gem5 Simulator	Chisel C++ RTL Simulator
KIPS	175	4

Source: Tuan Ta, et. al, "Simulating Multi-Core RISC-V Systems in gem5", [CARRV 2018]

Conclusion

- An infrastructure extension of Sniper
- Sniper + RISC-V is now available
- Next steps
 - Improve the simulator features to allow for a detailed comparison with cycle-level processor implementations

Alpha-version

- Thank you
- Download Today!
 - <http://snipersim.org/w/Download>
- Questions?
 - <http://groups.google.com/group/snipersim>



Flexible Timing Simulation of RISC-V Processors with Sniper

Neethu Bal Mallya¹, Cecilia Gonzalez-Alvarez², Trevor E. Carlson¹

¹National University of Singapore, Singapore

²Ghent University, Belgium

