QEMU-CAS: A Full-System Cycle-Accurate Simulation Framework based on QEMU

Ye Cao, Zhixuan Xu, Zhangxi Tan

RISC-V International Open-Source Laboratory (RIOS Lab)

Tsinghua University

OUTLINE

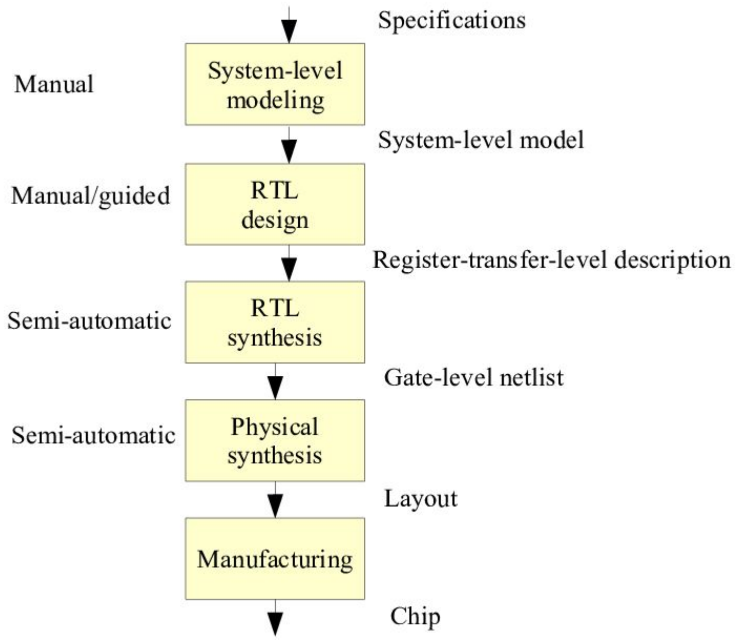
- Background
- Design Methodology
- Evaluation
- Conclusion

BACKGROUND

- Software Simulation
- Benchmarks
- Motivation

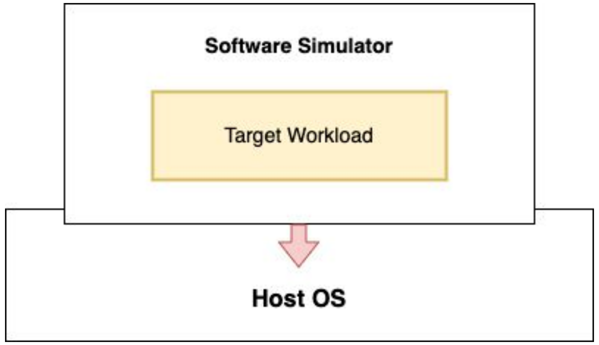
Hardware & Software Co-design

- Hardware development is a costly and time-consuming process
  - Validations are needed to ensure the quality and correctness



- **A typical hardware design flow[1]**

- Embracing hardware & software co-design
  - To reduce the cost and expense of design trial and error
  - Software simulation: modeling with software language on host OS

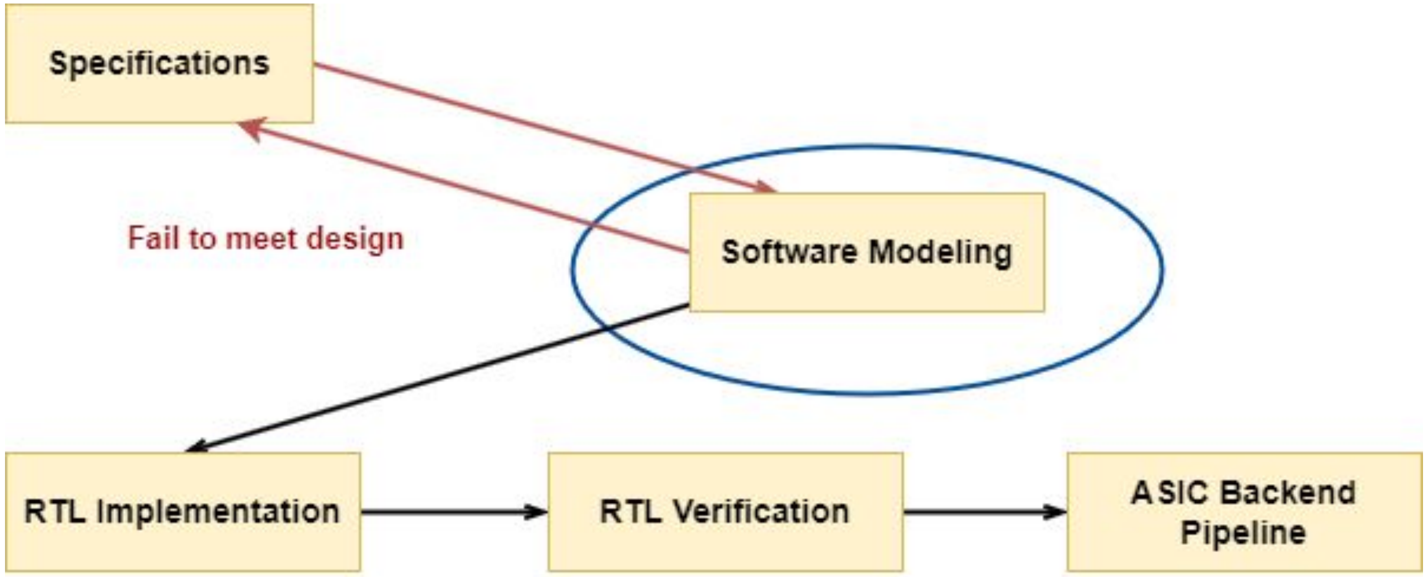


- **Software Simulators**

[1] Savaton G, Delatour J, Courtel K. Roll your own hardware description language[C]//OOPSLA & GPCE Workshop Best Practices for Model Driven Software Development. 2004.

Software Simulation

- A representative flow incorporating software modeling
  - Approaches to efficiently modifying and adapting the design
  - Accelerating the design process



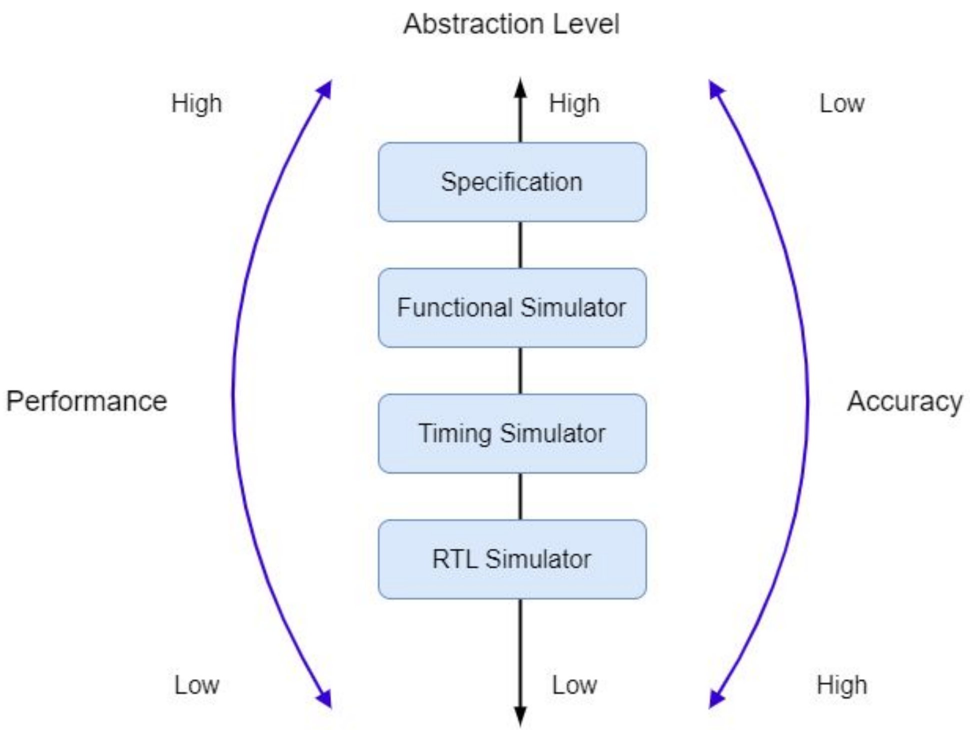
Benchmarks

- Benchmarks: important standards of measuring hardware capability and performance
  - Dhrystone, CoreMark, SPEC CPU, …
- Limitations of static, numeric benchmark
  - Mainly focus on some specific aspects, may not show the comprehensive picture
  - Unsensitive to emerging hardware & software technologies
- Dynamic benchmarks: real-world workloads & applications
  - Larger scale & complexity
  - Requirements for enhanced environment & peripherals

Simulator Classification

- Divided by abstraction levels
  - Functional simulator
  - Timing (performance) simulator

- Detailed timing implementation
  - Execution-driven simulator
  - Event-driven simulator



- **Abstraction levels of simulation**

Challenges & Motivation

- Existing challenges for performance simulators
  - Lack of capabilities to effectively simulate complex hardware components
  - Limited flexibility when dealing with large workloads

- What about functional simulators?
  - Capability & performance
  - Cannot provide architecture behaviors and details

- Research Objectives
  - Cycle-accurate simulation on advanced modern processor models
  - Capability of full system modeling
  - A dynamic approach to characterize the target workloads

QEMU-CAS

- A novel full-system cycle-accurate software simulation framework
  - Integrating a cycle-accurate CPU model with QEMU
  - Capable of modeling full-fledged Linux environments with modern IO peripherals

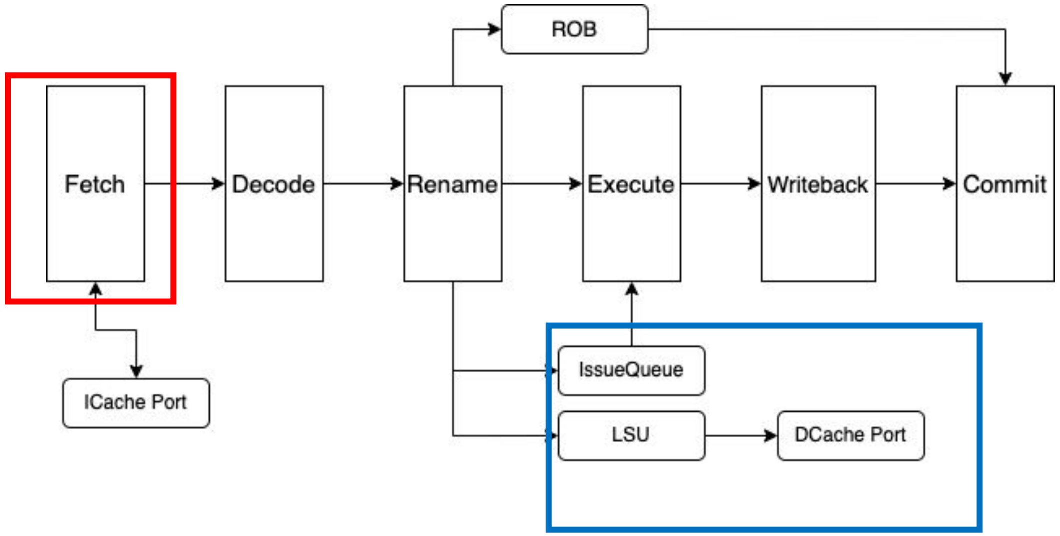
- Our Contributions
  - A simulation framework capable of modeling superscalar out-of-order processors
  - A novel methodology for full-system modeling and simulations on RISC-V
  - A dynamic switch-based mechanism to characterize the target workload
    - cycle-accurate performance analysis on a dynamic binary translation framework

DESIGN METHODOLOGY

- Components
- Design of QEMU-CAS
- CPU-IO IRQ Interface
- ISA Model
- Simulating Multi-Core Target Systems
- A Switch-based Approach to Characterizing Workload

Components

- Components: basic units of simulation
  - Stages: pipeline topology
    - the highest layer of the CPU model
    - maintain the timing and pipeline structure
  - Function Components: detailed functionalities
    - including data queues, function units, …

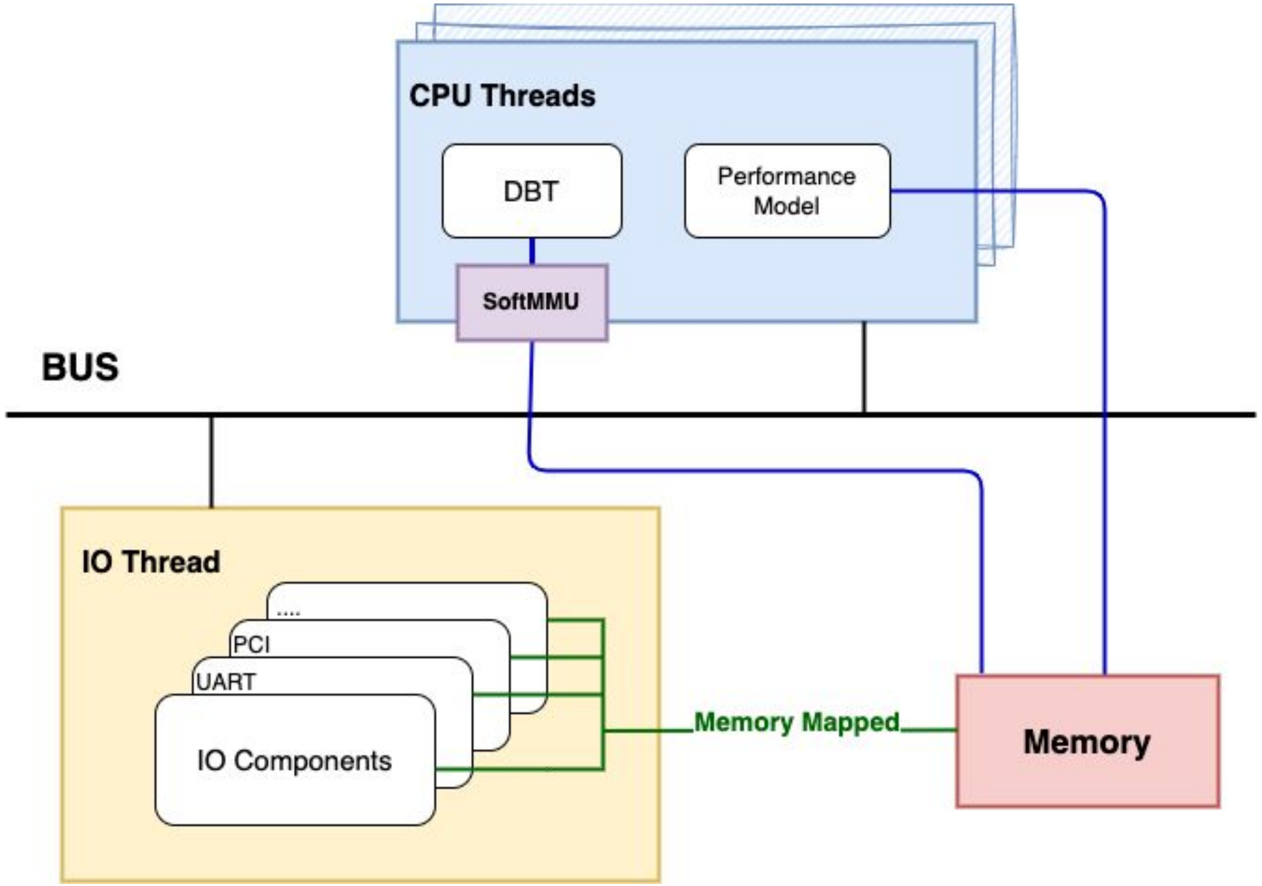


- **Pipeline of Gem5 Out-of-Order CPU Model**

Design of QEMU-CAS

- Multi-thread Structure
  - CPU Threads
  - IO Thread

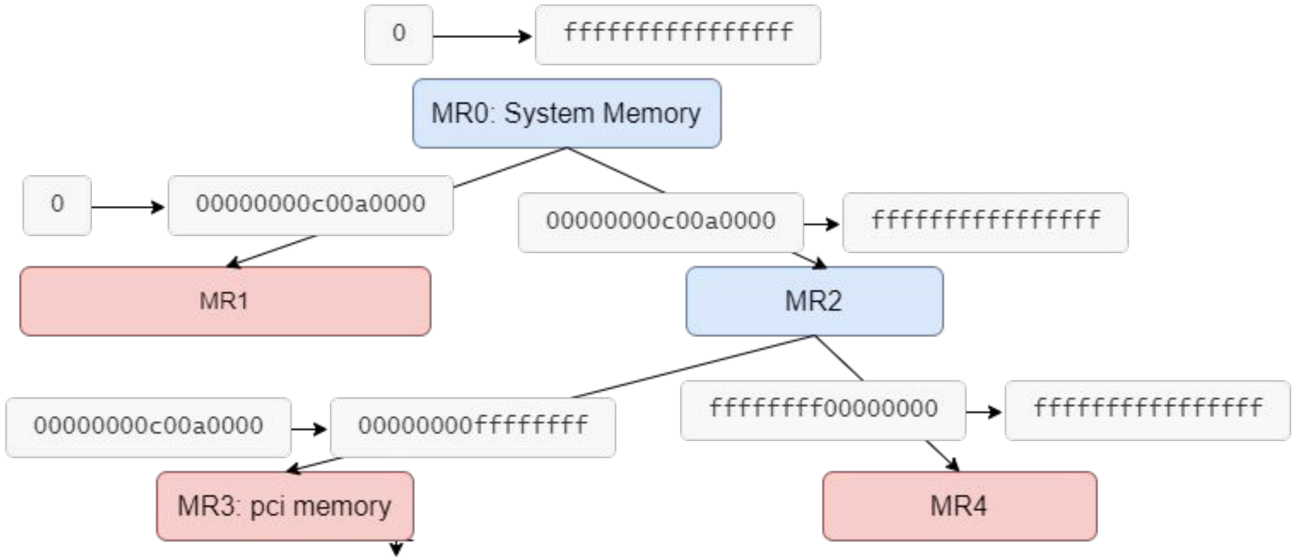
- Hybrid-driven Architecture
  - Execution-driven in performance CPU model
  - Event-driven for SoC simulation



- **Infrastructure of the simulator**

Memory Model

- Memory Region (MR)
  - contiguous range of memory that can be accessed by the CPU and IO

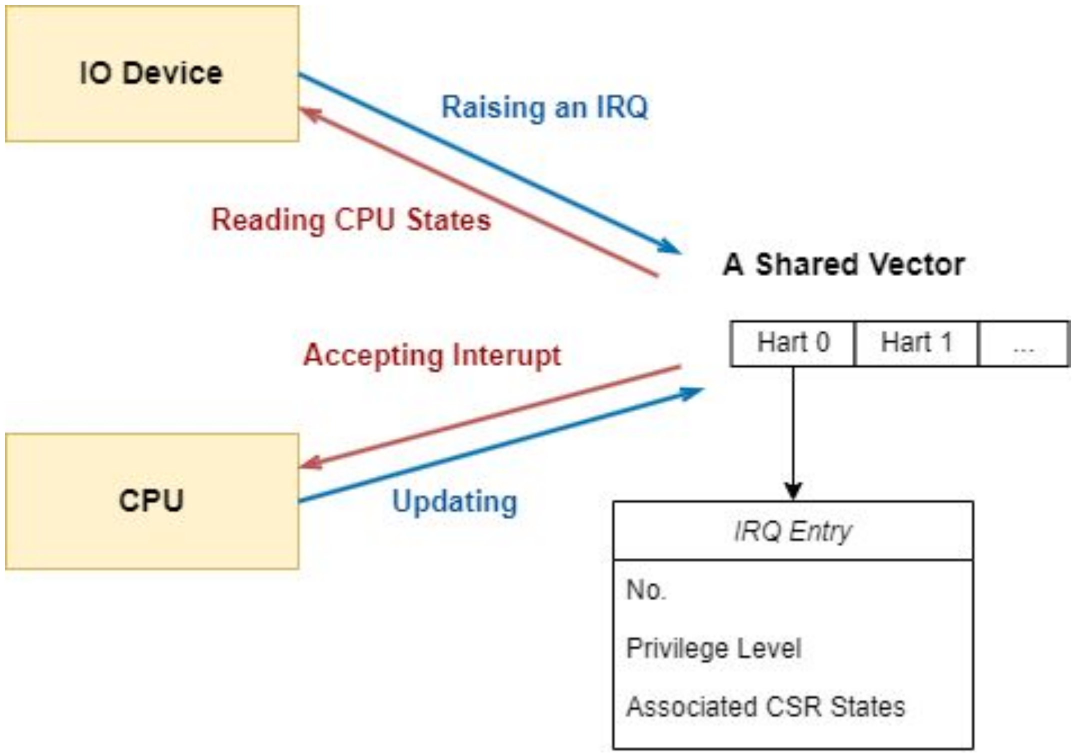


- **MR Tree**

- Easy for memory model customization
  - Decoupled from IO and CPU
  - Allowing timing models on top of MRs

CPU-IO IRQ Interface

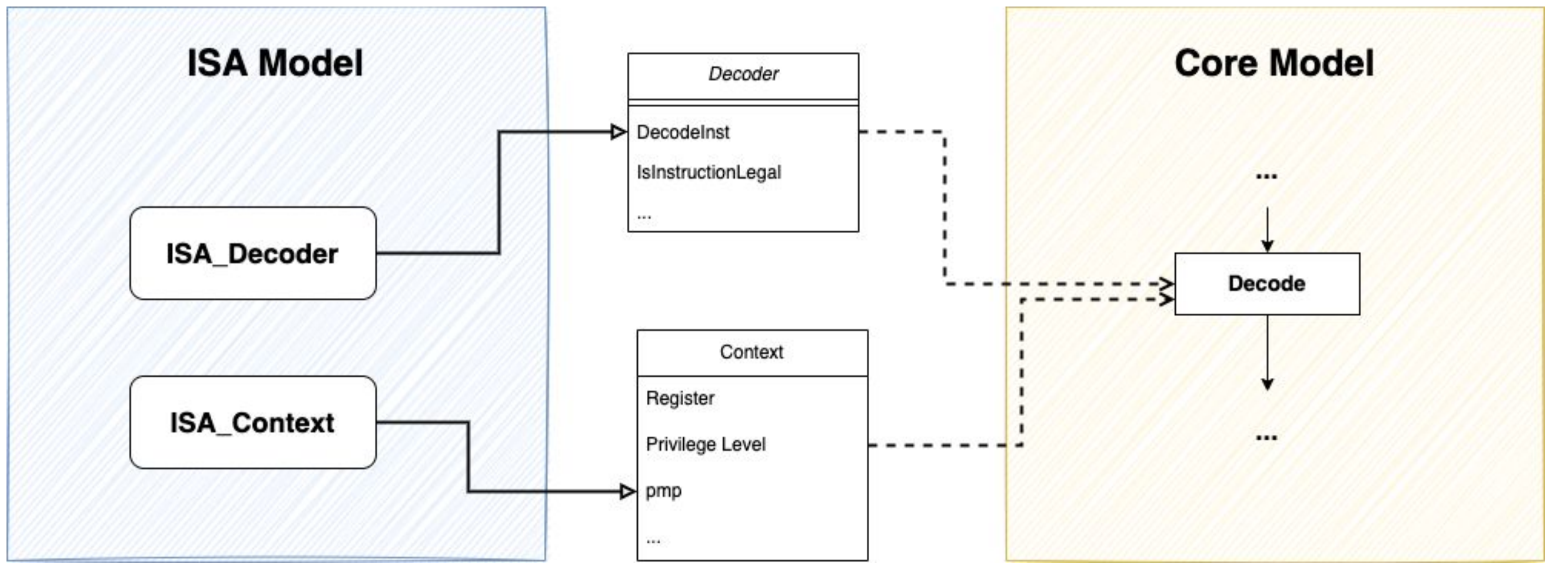
- A shared IRQ vector to keep the synchronization between different components



- **CPU and IO exchange IRQs through an IRQ Vector**

ISA Model

- The ISA Model is independent from CPU models

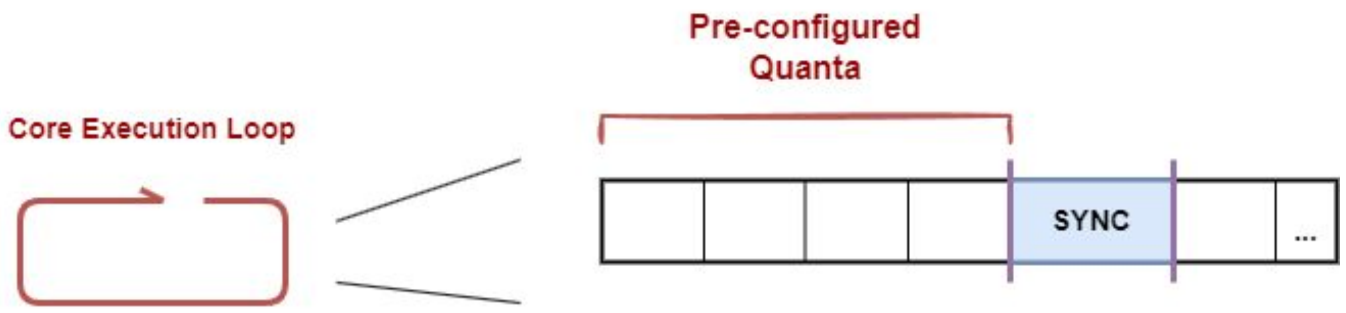


- **The ISA model is independent of the core model**

Discussion: Simulating Multi-Core Target Systems

- Our multi-thread architecture is suitable for simulating multi-core system
  - High concurrency in execution

- A key issue: trade-off of synchronization
  - Balancing the performance & accuracy of simulating multi-core system
  - A quanta-based approach



- **Core Execution with Quanta-based Synchronization**

A Switch-based Approach to Characterizing Workload

- For large-scale workloads, traditional snapshot methods have limitations
  - Limited Scalability issues
  - Unable to reflect real-time behaviors

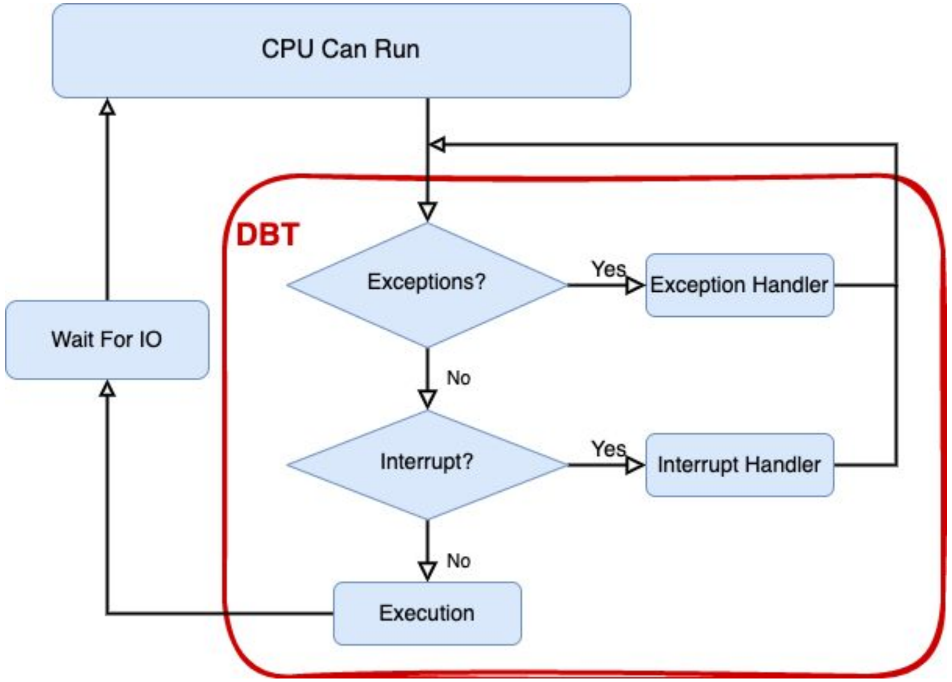
- A dynamic switch-based approach in QEMU-CAS
  - Capable of dynamically switching core models in runtime
    - A QEMU DBT & a performance core model
  - Sharing other hardware components in emulated SoC

Checkpointing Strategy

- Ideally, Current core needs stopping at the time it receives a switching command
  - In real case, the response may have a delay

    → may lead to inaccurate performance picture

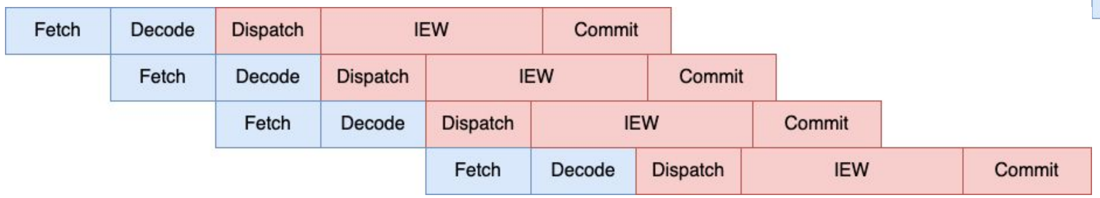
- Checkpointing on QEMU DBT
  - QEMU uses TBs (Translation Blocks) as basic units of instruction translation
  - No speculative operations cross TBs
  - A new exception with lowest priority
    - *EXCP_SWITCHING*



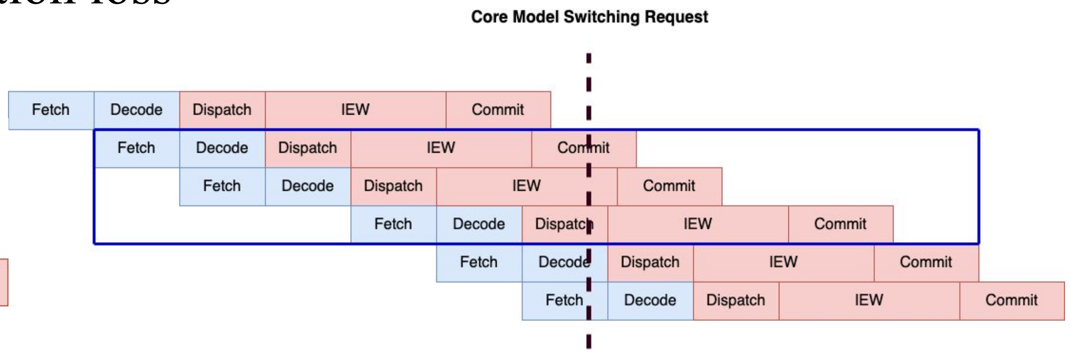
- **QEMU DBT workflow**

Checkpointing on Performance Model

- Instruction execution are not transactions in a performance core model
  - Instructions may take multiple cycles from fetched to committed
  - A simple switching method results in instruction loss



- **An out-of-order pipeline**



- **Instructions in blue square are lost when switching**

- Uncommitted instructions should not affect the architecture states
  - None instructions are at backend.
  - At least one fetched instruction in the frontend
  - No pending interrupts in pipeline

Time Dilation

- Simulated Time vs Real Time
  - Normally, they need to tick at the same time; But in some cases, simulated time need a dilation

- Timer
  - $TS(simulated) = TS(rt) * basefreq + delta$

- Clock Alignment in Core Switch
  - CPU frequency writes to FDT when system booting
  - Performance model cannot run as fast as QEMU
  - $delta_{new} = TS_{rt} * (basefreq_{old} - basefreq_{new}) + delta_{old}$

EVALUATION

- Experiment Setup
- Simulator Performance
- Case Studies

Experiment Setup

- Fedora Linux in RISC-V
  - OS Image: uboot + Fedora Rawhide

- Core Configurations

Table 1: Parameters of core models

Item	Configuration
Fetch Width	8
ROB Entry Size	32
Load/Store Queue Entry	16
TLB Entry	8
BTB Size	1024
RAS Size	16
Tournament - Local Predictor Size	1024
Tournament - Local History Entry Bits	10
Tournament - Global History Entry Bits	12
Physical Register File	128

Simulator Performance

- In the simulation, QEMU DBT runs much faster than performance model
  - about 1000x
- Changing the inner system design causes performance change to original model
  - Taking BPU as an example, removing it results in 18% performance loss
  - far less difference when compared to DBT performances

Table 2: Performance of DBT and core model

Simulator	Time per cycle(μs)
QEMU DBT	0.241
Performance Model	249.024
Performance Model, no BPU	294.363

Case Study: Linux Ping Test

- Ping is a commonly-used tool to test the reachability to Internet
  - can be used as a benchmark in hardware development to test the capability of networking

Algorithm 5.1 run_ping_test.sh

Arguments: numbers of ping n
target website url
output path $output\_path$

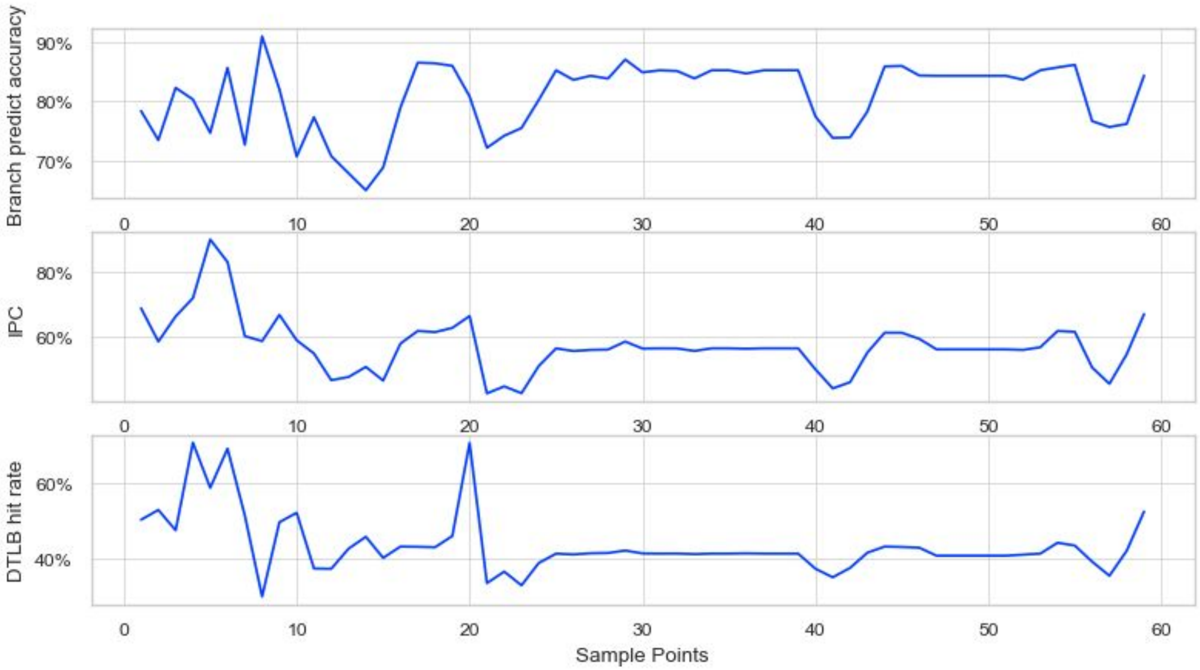
./prof_tool begin

ping -c n url > $output\_path$

./prof_tool terminate

- Redirect to a text file
  - to test the support of block devices

- Sampling per 0.1 million cycles

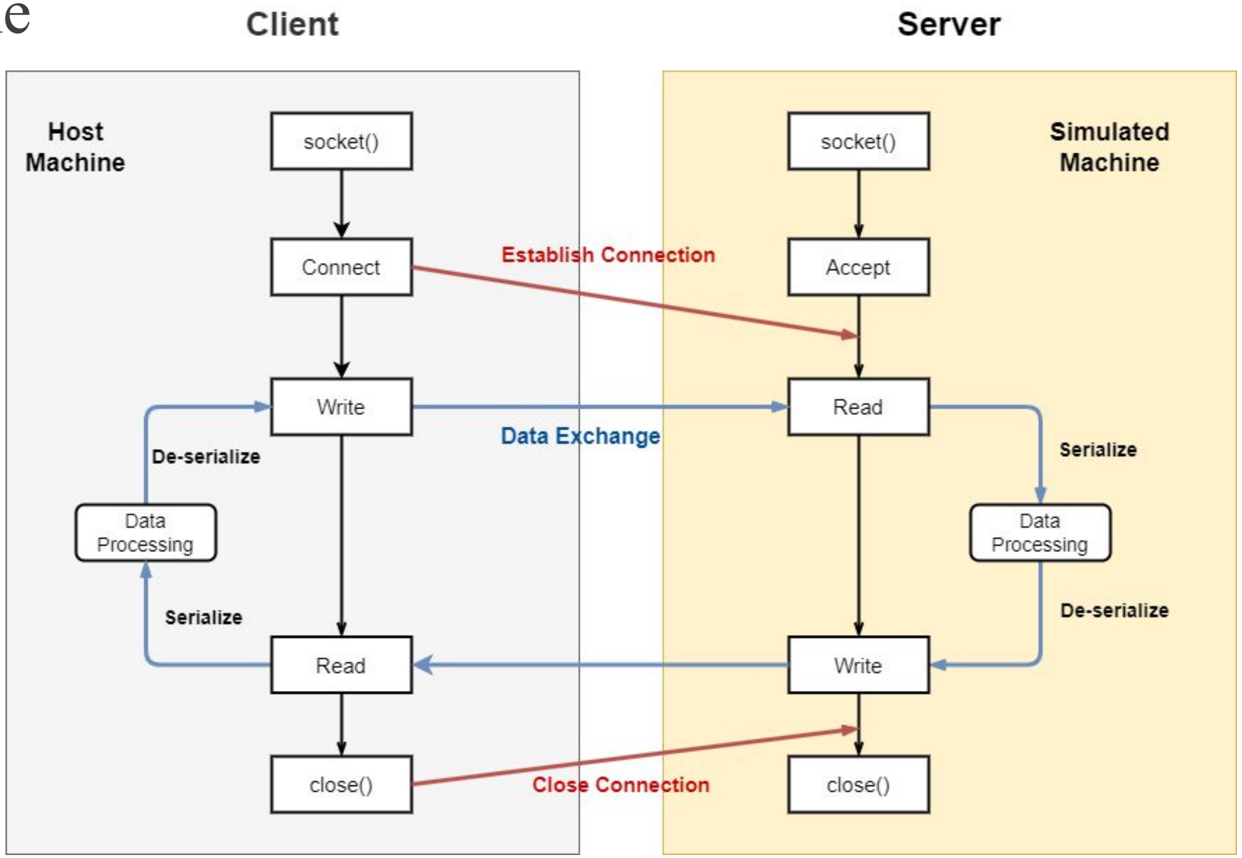


- **PMU result of the execution**

Case Study: Socket Communication

- We conduct an experiment to investigate the hardware behavior in a socket network
  - building a client/server framework
    - server.c: run on a simulated OS
    - client.c: run on host machine
  - focusing on the segments of data processing

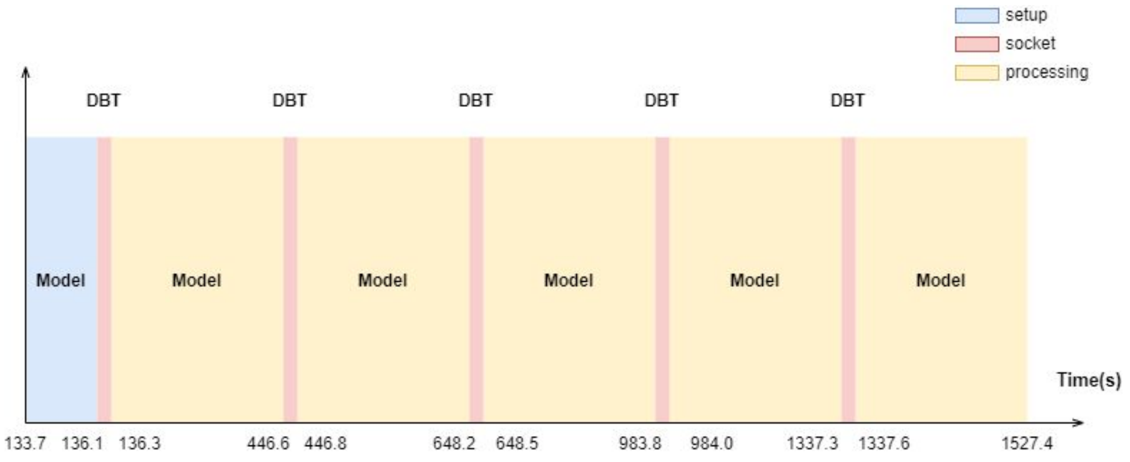
- Hard for traditional snapshot approach to locate the target program segments
  - Scalability issues
  - Static approaches are hard to solve real-time communications



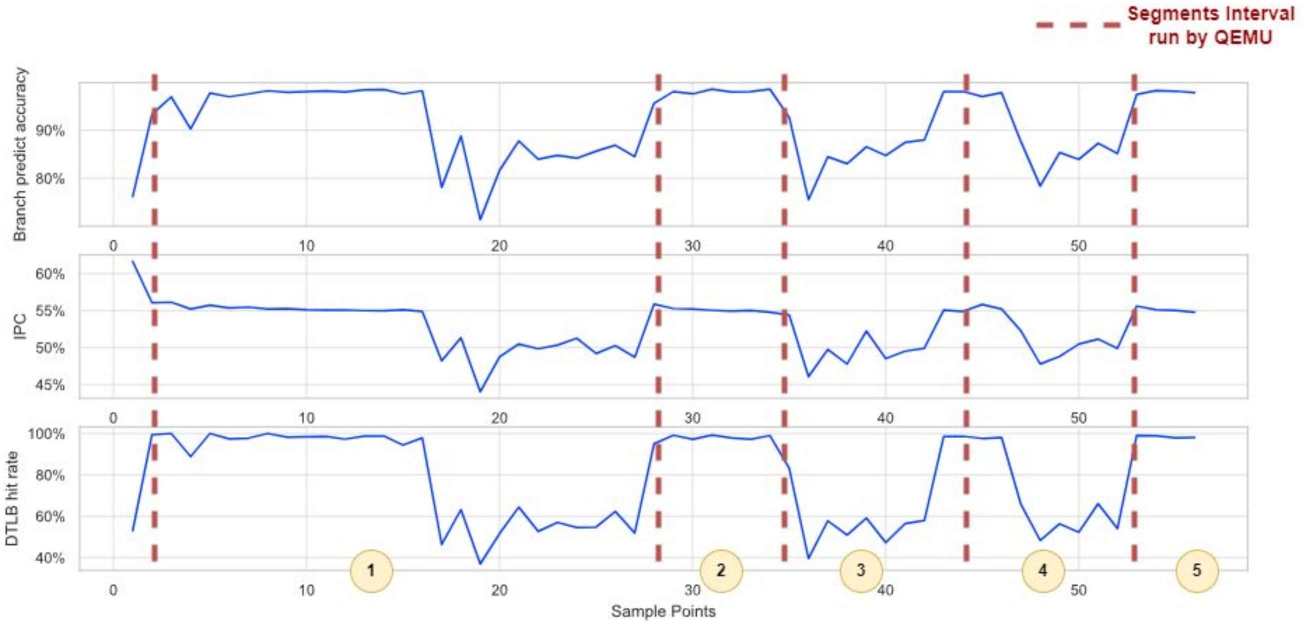
- A client/server framework used in the experiment

Case Study: Socket Communication(2)

- Dynamic model switching can accurately characterize the workload
  - run data processing on performance model
  - switch to QEMU DBT at socket communication



- **Time Spent on performance model & QEMU DBT**



- **PMU result of the execution**

CONCLUSION

Conclusion

- We build a hybrid-driven and full-system simulator adopted from QEMU
- We propose a dynamic switch-based approach to characterizing large workload
- We use several real-world applications to evaluate the capability and performance of our simulation framework.

Future Work

- Enhanced implementation on multi-core systems

- Compatibility with other memory models