XUANTIE

CPU Utilization Counters for RISC-V

An Architectural Extension for CPU Utilization Tracking on RISC-V

Esther Zhang

Alibaba Damo Academy
Esther.Z@linux.alibaba.com

The Problem: Scheduling Blindness

Why CPU load metrics are fundamentally incomplete?

❌ TRADITIONAL LOAD

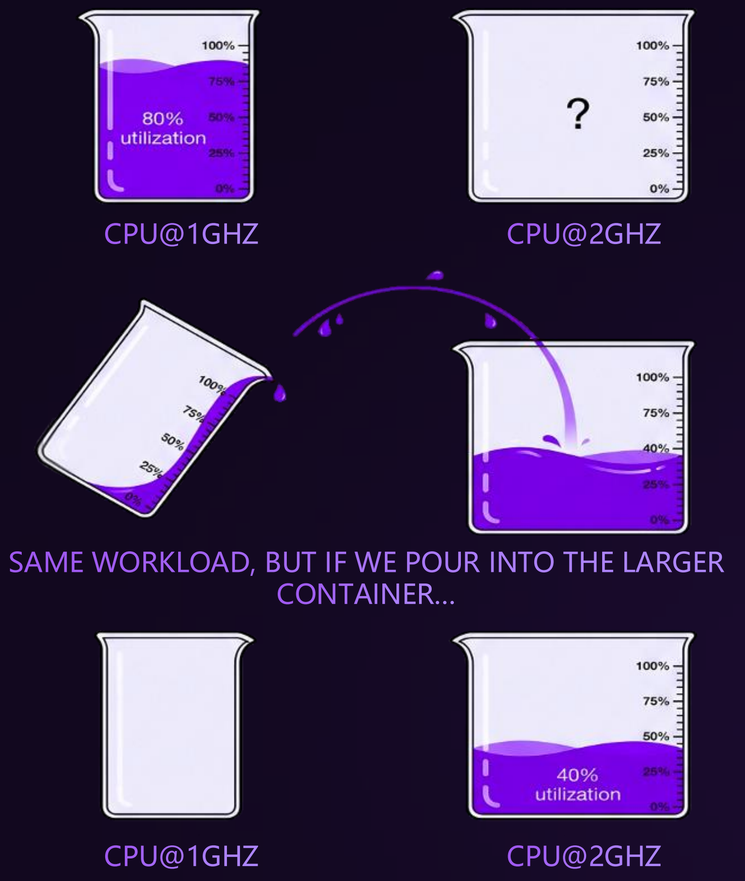
A RISC-V CPU reports **80%** utilization.
The scheduler decides: "No need to migrate tasks. No need to boost frequency."

✅ TRUE LOAD

But the CPU is actually running at only **50%** of max frequency.
The real computational load is closer to **40%**, not 80%.

💡IMPACT

Result: Wrong task placement, wasted power, and missed performance opportunities.



PELT & Frequency Invariance

01 What PELT Does

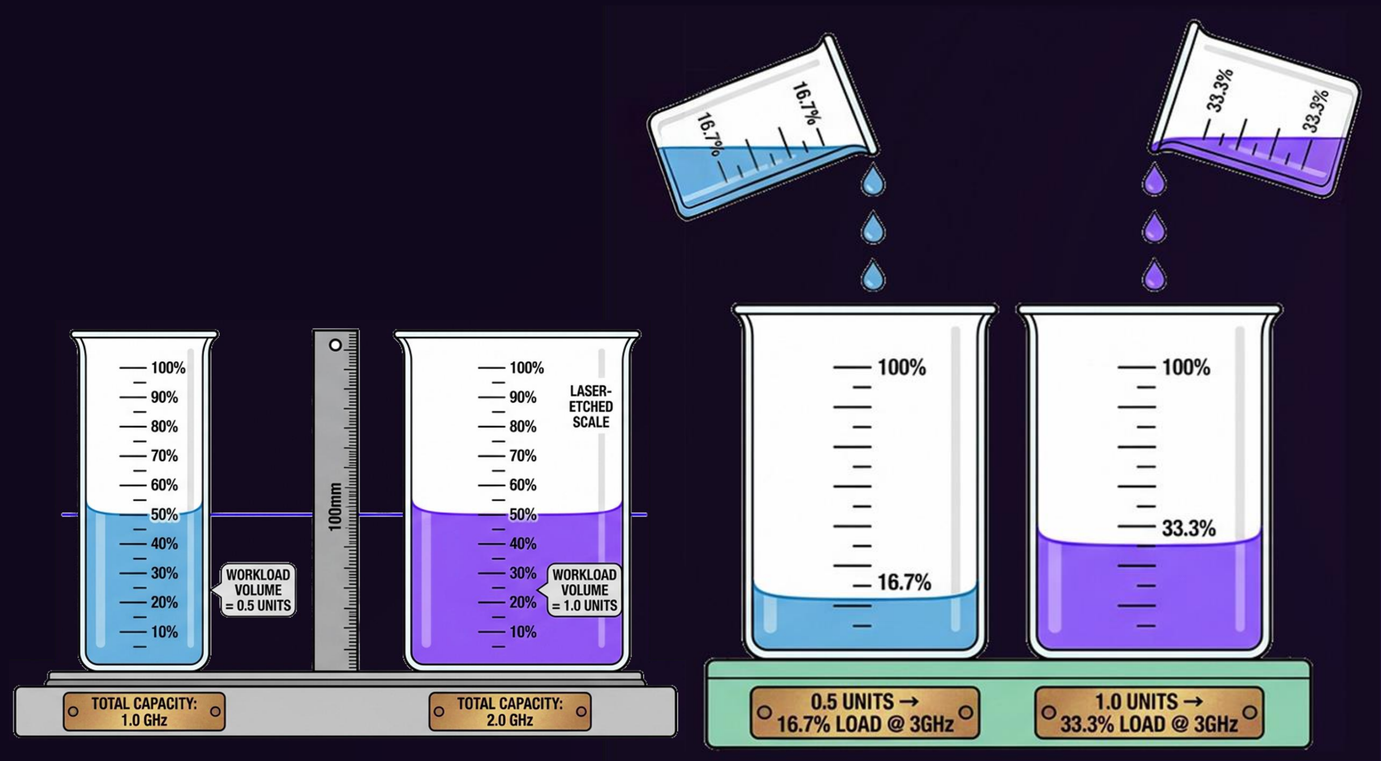
- Linux scheduler tracks per-entity load via exponential weighted moving average
- Produces **util_avg**: a smoothed metric of task demand

$$\text{util\_avg} \approx \frac{\textit{weighted total time}}{\textit{weighted running time}} \in [0, 1024]$$

02 The DVFS distortion

- Dynamic Voltage & Frequency Scaling changes clock rate
- At lower freq, same work takes more wall-clock time
- **util_avg inflates** artificially without correction

Frequency Invariance scales observed time by $\frac{f\_cur}{f\_max}$



The Gap: RISC-V Has no Standard Solution

XUANTIE

X86-64	ARM v8.4+	RISC-V
APERF/MPERF	**AMU**	**?**
Hardware counters for actual vs. reference cycles	Architectural counters frequency invariance	No standard mechanism

The Proposal: Two Architectural Counters

XUANTIE

We introduce read-only 64-bit counters that are unaffected by mcounterinhibit and pause in idle states:

mcorecyc

CPU Active Cycle Counter

- Increments at the current CPU frequency
- Pauses when CPU enters idle (WFI)
- Not affected by mcounterinhibit

mactime

CPU Active Time Counter

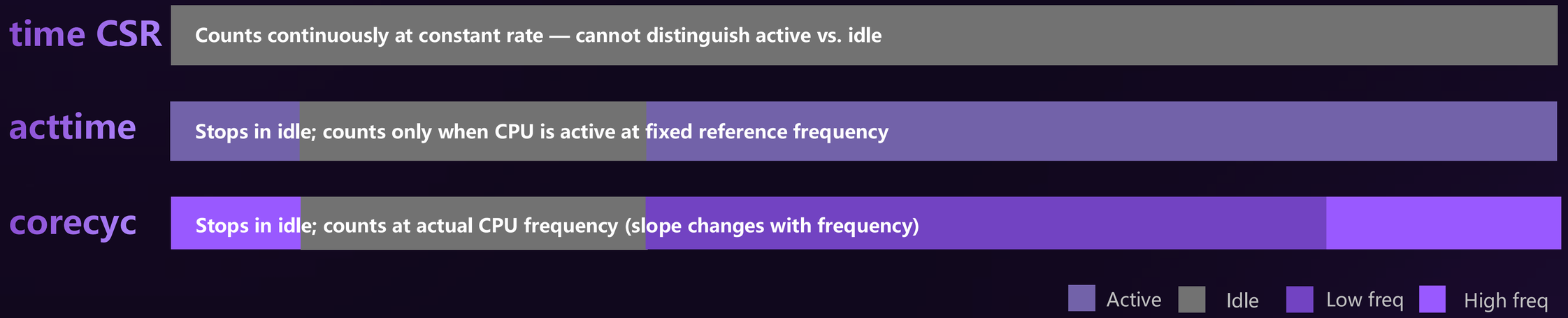
- Increments at a fixed reference frequency
- Pauses when CPU enters idle (WFI)
- Derived from same timebase as time CSR

Both counters initialize to zero on reset and wrap around on overflow without generating interrupts.

How it Works: A Visual Comparison

XUANTIE

Timeline: Active → Idle (WFI) → Active at lower freq → Active at higher freq



The ratio of corecyc to acttime reveals the effective running frequency.

The Math: Deriving Frequency & Utilization

XUANTIE

Effective Running Frequency

f_running = f_fixed * (delta_corecyc / delta_acttime)

- delta_corecyc = corecyc increment over observation window
- delta_acttime = acttime increment over same window
- f_fixed = constant reference frequency

Normalized Utilization (for PELT Frequency Invariance)

normalized_util = (f_fixed / f_max) * (delta_corecyc / delta_time)

- delta_time = architectural time CSR increment (wall-clock)
- f_max = maximum operating frequency

Both formulas use pure CSR reads — no traps, no mailbox queries, no firmware calls.

Why Existing Counters Are Insufficient?

XUANTIE

01

time CSR counts wall-clock time continuously

It keeps incrementing even when the CPU is idle (WFI).
— The scheduler needs active compute time, not wall time.

02

mcycle conflicts with performance profiling tools

perf dynamically enables/disables `mcycle` via `mcountinhibit`, requiring traps to M-mode.
Scheduler hot-path updates occur at kHz-level context switch rates
— **Races** and **overhead** are unacceptable.

03

Software frequency queries are slow and stale

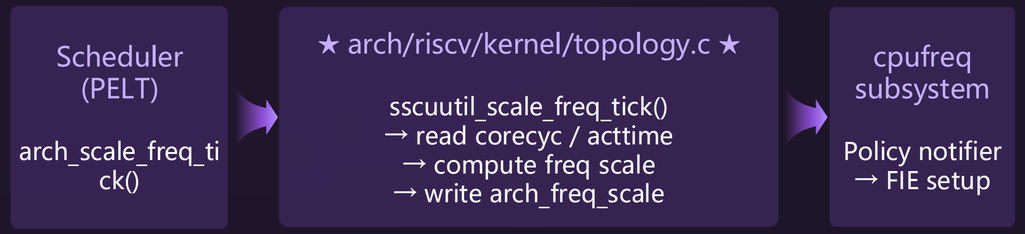
Modern SoCs delegate final frequency decisions to a hardware Power Controller (PuC).
— The AP must query via mailbox/SMC, introducing **latency** and **stale data**.

Proof of Concept

Based on pre-ratification spec - implementation details subject to change

Linux Kernel

- Runtime ISA detection (Sscpuutil)
- FIE integration via topology_set_scale_freq_source()
- cpufreq policy notifier for per-CPU setup
- debugfs: /sys/kernel/debug/sscpuutil_fie



https://github.com/XUANTIE-RV/linux/pull/26

OpenSBI

- Enable counter access via CSR configuration

https://github.com/XUANTIE-RV/opensbi/pull/11

Qemu

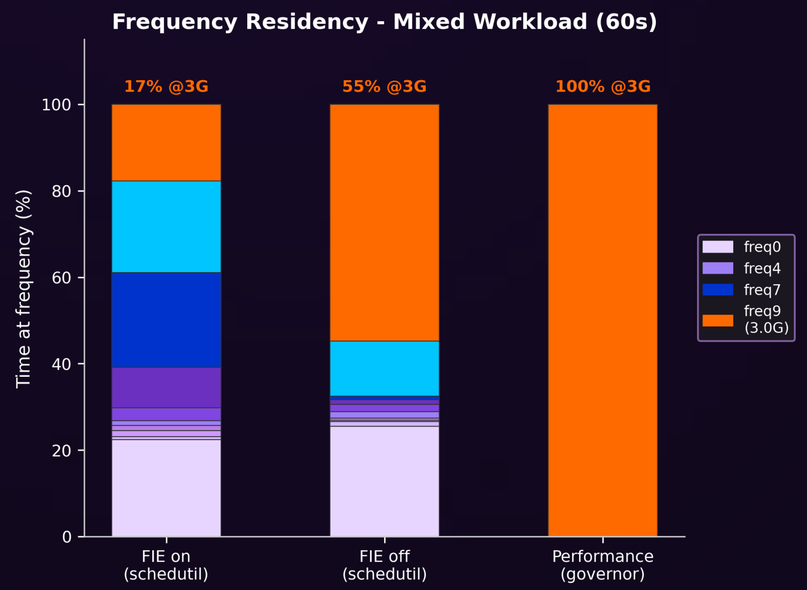
- Emulates mcorecyc /macttime CSRs with frequency scaling and idle pause

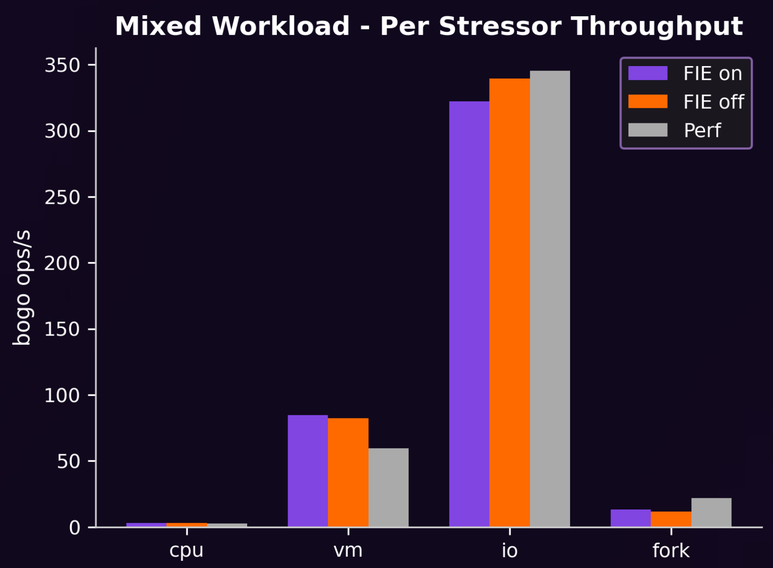
https://github.com/XUANTIE-RV/qemu/pull/13

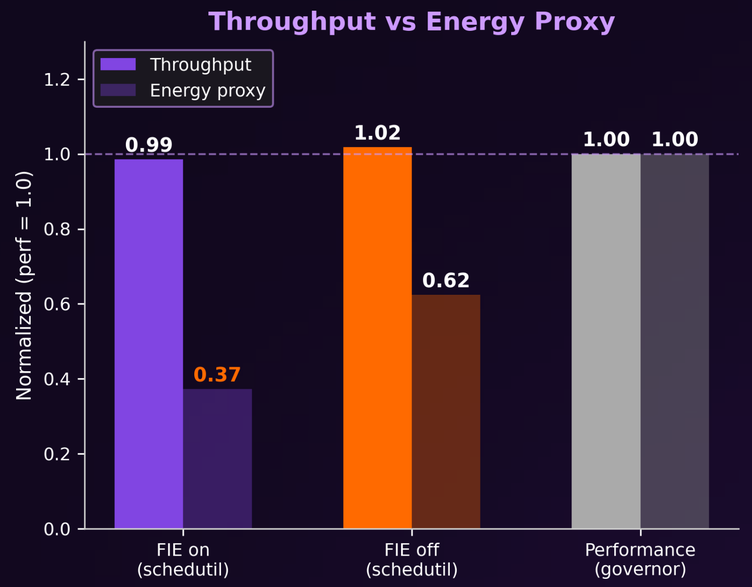
✅ Validated on Real Silicon

Sscuutil Hardware: Benchmark Results

XuanTie C950 8-core, Linux 6.6.36







Key Results (XuanTie C950 8-core)

- **Mixed Workload (cpu + vm + io + fork):**

FIE on: 99% throughput, **37%** energy **(↓63%)**

FIE off: 102% throughput, **62%** energy **(↓38%)**

Perf: 100% throughput, **100%** energy (baseline)

Conclusion:

Sscpuutil FIE enables accurate PELT load tracking → schedutil selects optimal OPP → 63% energy reduction with **<1%** performance loss

Why This Design Is the Right Fit

XUANTIE

Minimal Hardware Cost

Two 64-bit counters + clock gating logic.

No new state machines

No trap handlers.

Scheduler Hot-Path Friendly

Pure CSR reads at any privilege level.

No M-mode traps.

No mcountinhibit races.

Dedicated & Isolated

Separate from mcycle /HPM

No conflict with perf tools

Not affected by mcountinhibit

Linux-Ready

Maps directly to existing arch_scale_cpu_capacity() infrastructure.

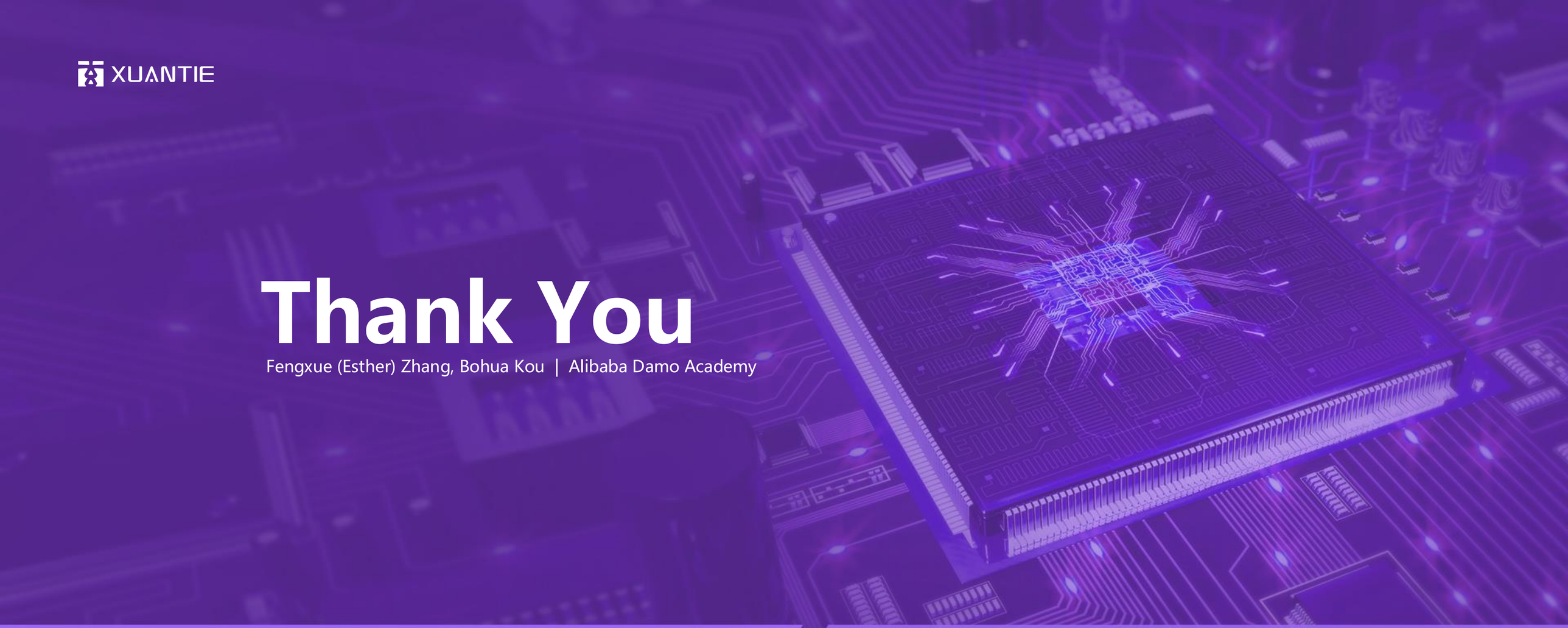
No kernel ABI changes required.

Role separation: PMU/Sscofpmf handles performance profiling. cpuutil handles OS power management. Each tool stays in its lane.

XUANTIE

Thank You

Fengxue (Esther) Zhang, Bohua Kou | Alibaba Damo Academy



玄铁公众号

玄铁官网

公司地址：浙江省杭州市余杭区阿里巴巴西溪园区C区
如需了解更多，可以咨询您的【玄铁专家】
您也可前往玄铁官网：xrvm.cn 或通过 xuantie@service.alibaba.com，
riscv_techsales@service.alibaba.com 与我们联系

XUANTIE

CSR Architecture

CSR Name	Privilege	Purpose
mcorecyc / mcorecych	Machine	Active cycle counter (lower / upper 32b for RV32)
macttime / macttimeh	Machine	Active time counter at fixed freq (lower / upper 32b)
mcpuutilen	Machine	Visibility control to next lower privilege level
corecyc / corecych	Supervisor / User	Shadow of mcorecyc (when enabled)
acttime / acttimeh	Supervisor / User	Shadow of macttime (when enabled)
hcpuutilen	Hypervisor	Guest OS visibility control (optional)

Consistent with RISC-V's existing time/mtimecmp philosophy: read-only counters with layered privilege gating.