RISC-V Summit China 2024

Enabling Hardware Sampling Based PGO for RISC-V Platform

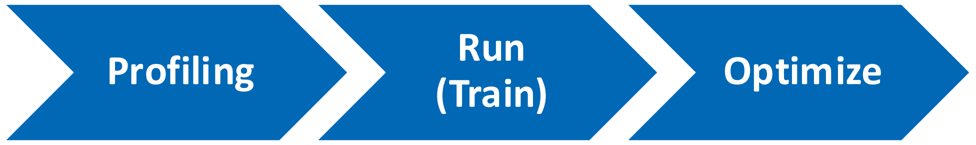
Gao Yichuan (yichuan.gao@intel.com)

RISC-V Agile Design Lab

Intel Labs China

intel®

PGO Basics



- PGO (Profile-Guided Optimization)
  - Use runtime feedback to improve software performance

- Software vs Hardware PGO

SWPGO (Instrumentation)

- Insert profiling code snippet to software
- Profiled data:
  - Branch/Jump count/target
  - Register/Memory values
- No hardware requirement
- High profiling overhead

HWPGO (Sampling)

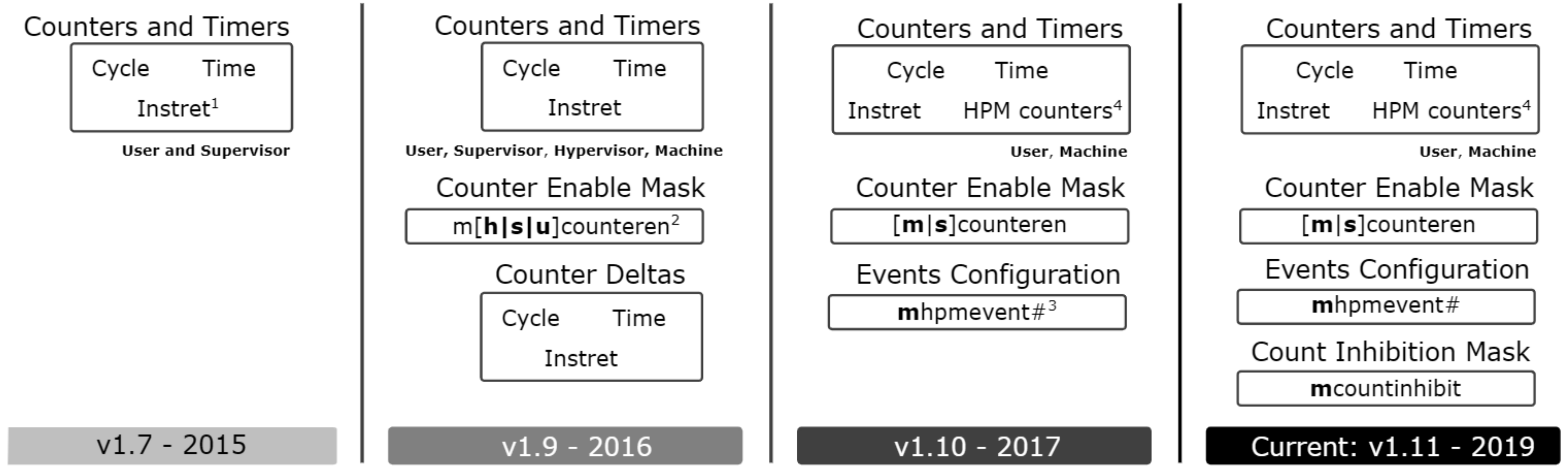
This talk

- Use hardware sample collect methods
- Profiled data:
  - Performance Counter values
  - Branch/Jump count/target (with LBR/CTR)
  - Register/Memory values (not yet)
- Require PMU hardware support
- Low profiling overhead

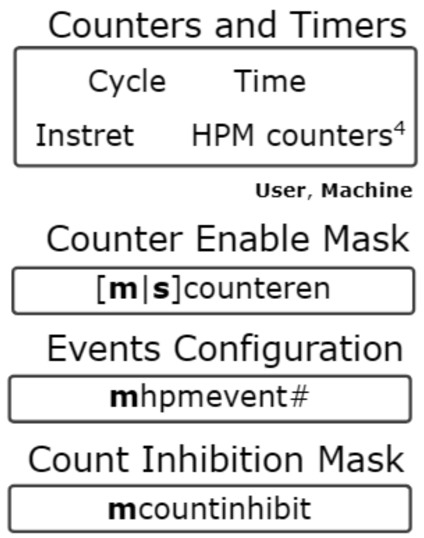
RISC-V PMU

- PMU (Performance Monitoring Unit)
  - Hardware unit for counting occurrence of uArch events

- RISC-V PMU: provide fixed and event-based hpmcounters
  - Software access via CSR interface



RISC-V PMU Extensions for HWPGO



- Base extensions (widely adopted)
  - Zicntr & Zihpm
    Provide basic counters for PMU events
- Latest PMU extensions
  - Smcdeleg & Ssccfg
    PMU counter delegation

    PMU config in S-mode
  - Smcntrpmf & Sscofpmf
    Counter mode filtering and overflow interrupt

    Event-based sampling

    RVA23
  - Smctr & Ssctr
    Hart control transfer records (in next slide)

    Control flow info recording
  - Future extensions for precise event sampling, register value profiling, etc.

RISC-V CTR for HWPGO

- Record control transfer history
  - Jump instructions (including function calls and returns)
  - Taken/not-taken branch instructions
  - Traps, and trap returns

- Data is organized as circular buffer (FIFO)
  - Source PC, Target PC
  - Type, Cycle Count
  - Misprediction

- Support filtering by transfer type / privilege mode
- Low overhead sampling

RISC-V Control Transfer Records (Smctr/Ssctr)

RISC-V Control Transfer Records TG

Version v1.0_rc3, 2024-07-24: Frozen

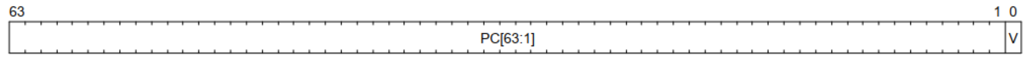


Figure 5. Control Transfer Record Source Register Format for MXLEN=64

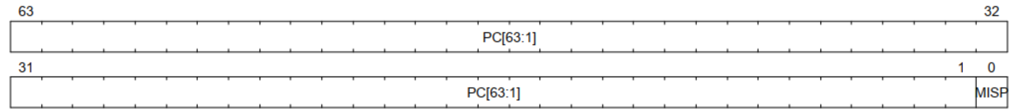


Figure 6. Control Transfer Record Target Register Format for MXLEN=64

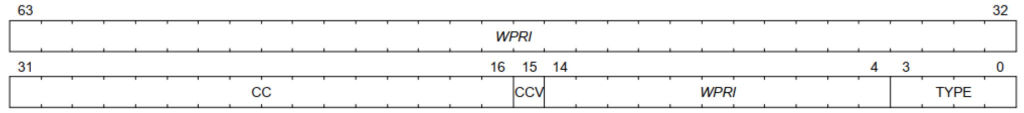
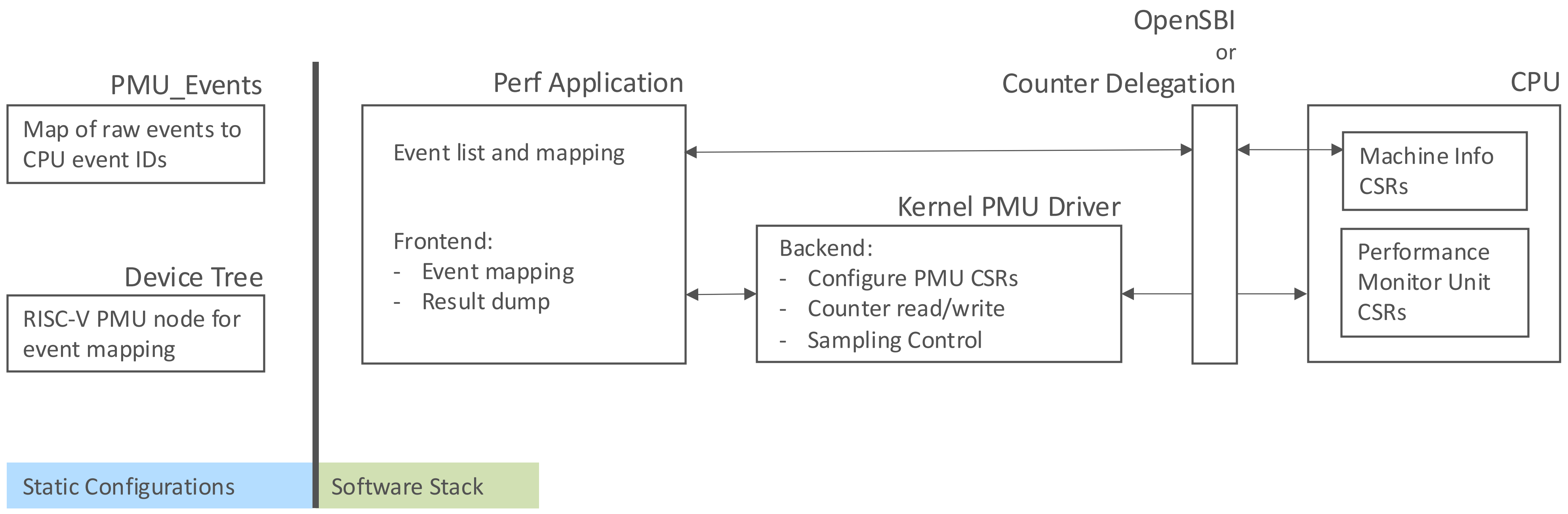


Figure 7. Control Transfer Record Metadata Register Format

Specifications: https://github.com/riscv/riscv-control-transfer-records/

RISC-V Perf Software



Case Study: Branch Prediction Profiling

- Perf tool sampling with CTR info

```
$ perf record -e branches:upp -e branch-misses:upp -b -z3 -- ./unpredictable
...
[ perf record: Woken up 4033 times to write data ]
[ perf record: Captured and wrote 8.454 MB perf.data, compressed (original
1008.905 MB, ratio is 120.537) ]
```

```
$ perf report -i perf.data --header-only
# data offset    : 424
# data size      : 8864755
# feat offset    : 8865179
# hostname : buildroot
# os release : 6.9.0
# perf version : 6.9.0
# arch : riscv64
# cmdline : /usr/bin/perf record -e branches:upp -e branch-misses:upp -b -z3 -- ./unpredictable
# event : name = branches:upp, , id = { 12 }, type = 0 (PERF_TYPE_HARDWARE), size = 136, config = 0x4
(PERF_COUNT_HW_BRANCH_INSTRUCTIONS), { sample_period, sample_freq } = 4000, sample_type = IP|TID|TIME|ID|PE>
# event : name = branch-misses:upp, , id = { 13 }, type = 0 (PERF_TYPE_HARDWARE), size = 136, config = 0x5
(PERF_COUNT_HW_BRANCH_MISSES), { sample_period, sample_freq } = 4000, sample_type = IP|TID|TIME|ID|PER>
# contains samples with branch stack
```

Demo application source code (Intel SATG) https://github.com/tcreech-intel/hwpgo-mispredict-example
Experimental CTR kernel drivers (community) https://github.com/rajnesh-kanwal/linux/tree/ctr_upstream_v2

Case Study: Mispredict Profile Feedback

```
for (int i = 0; i < N; i++) {
  int *p;
  if(s1[i] > 8000) {
    p = &s2[i];
    int z = i * i * i * i * i * i * i;
    nop(p, z);
  } else {
    p =  &s3[i];
    nop(p, 3);
  }
  dst[i] = *p;
}
```

Poorly predicted from perf events:

- **branches:upp**
- **branch-misses:upp**

Attach 'unpredictable' Metadata to BR instruction

HWPGO

```
for (int i = 0; i < N; i++) {
  int *p;

  int c = (s1[i] > 8000);
  p = c ? (&s2[i]) : (&s3[i]);
  int z = i * i * i * i * i * i * i;
  int arg2 = c ? z : 3;
  nop(p, arg2);

  dst[i] = *p;
}
```

Branch Predication

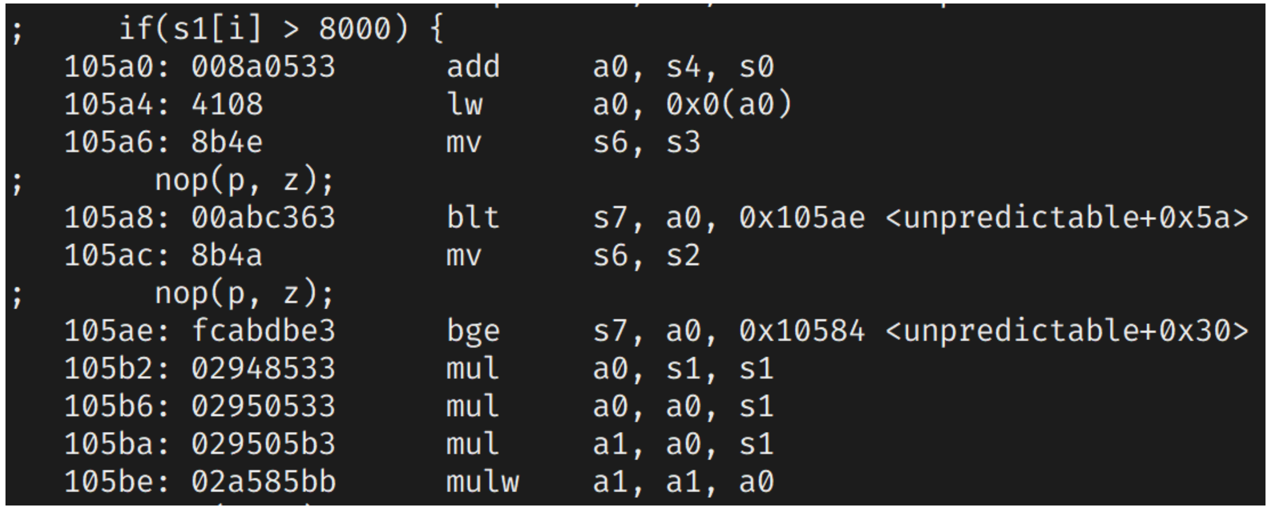
```
$ llvm-profgen --binary=unpredictable --output=unpredictable.prof --perfdata=perf.data --format=text
```

```
$ llvm-profgen --binary=unpredictable --output=unpredictable.misp.prof --perfdata=perf.data.4 --
format=text --perf-event=branch-misses:upp --leading-ip-only
```

Mispredict profile generate (from perf.data) with llvm-profgen *

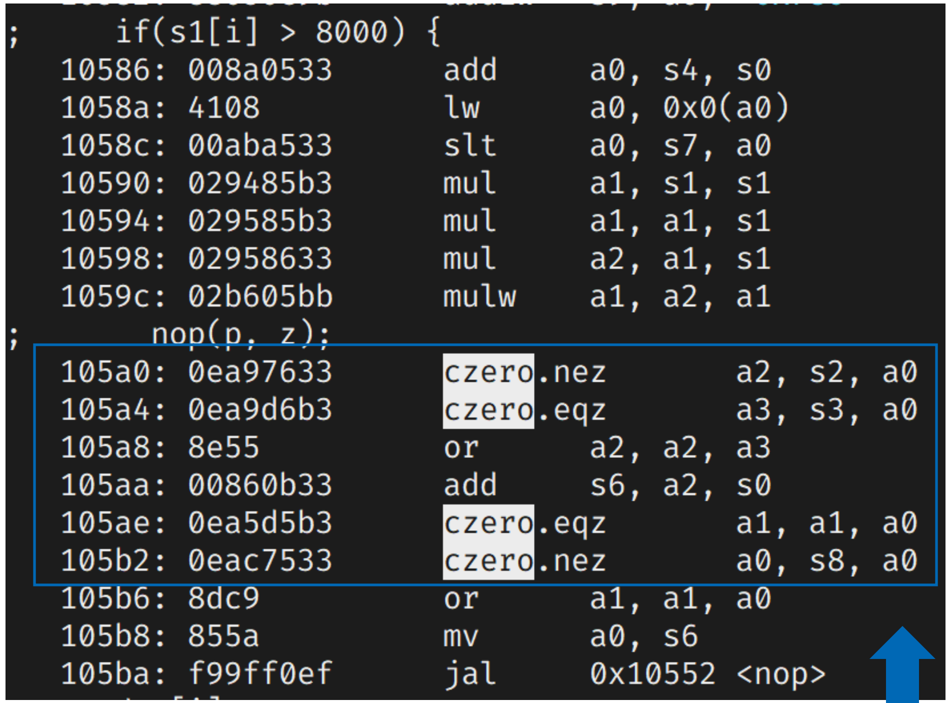
* LLVM codebase upstreaming in progess

Case Study: Optimize with Clang



```
;     if(s1[i] > 8000) {
   105a0: 008a0533       add     a0, s4, s0
   105a4: 4108           lw      a0, 0x0(a0)
   105a6: 8b4e           mv      s6, s3
;         nop(p, z);
   105a8: 00abc363       blt     s7, a0, 0x105ae <unpredictable+0x5a>
   105ac: 8b4a           mv      s6, s2
;         nop(p, z);
   105ae: fcabdbe3       bge     s7, a0, 0x10584 <unpredictable+0x30>
   105b2: 02948533       mul     a0, s1, s1
   105b6: 02950533       mul     a0, a0, s1
   105ba: 029505b3       mul     a1, a0, s1
   105be: 02a585bb       mulw    a1, a1, a0
```

Before HWPGO



```
;     if(s1[i] > 8000) {
   10586: 008a0533       add     a0, s4, s0
   1058a: 4108           lw      a0, 0x0(a0)
   1058c: 00aba533       slt     a0, s7, a0
   10590: 029485b3       mul     a1, s1, s1
   10594: 029585b3       mul     a1, a1, s1
   10598: 02958633       mul     a2, a1, s1
   1059c: 02b605bb       mulw    a1, a2, a1
;         nop(p, z);
   105a0: 0ea97633       czero.nez         a2, s2, a0
   105a4: 0ea9d6b3       czero.eqz         a3, s3, a0
   105a8: 8e55           or      a2, a2, a3
   105aa: 00860b33       add     s6, a2, s0
   105ae: 0ea5d5b3       czero.eqz         a1, a1, a0
   105b2: 0eac7533       czero.nez         a0, s8, a0
   105b6: 8dc9           or      a1, a1, a0
   105b8: 855a           mv      a0, s6
   105ba: f99ff0ef       jal     0x10552 <nop>
```

After HWPGO

Unpredictable branch instruction eliminated!

Predication using RISC-V Zicond instruction

```
$ clang -march=rv64gc_zba_zbb_zbc_zbs_zicond -mabi=lp64d --target=riscv64-unknown-linux-gnu --
sysroot=riscv64-chroot -O3 -static -std=gnu99 nop.c unpredictable.c -fprofile-sample-
use=unpredictable.prof -mllvm -unpredictable-hints-file=unpredictable.misp.prof -o unpredictable_pgo
```

Case Study: Performance Results

```
# perf stat -- ./unpredictable

 Performance counter stats for './unpredictable':

  178950.65 msec task-clock
         97      page-faults
 2885889754      cycles
 3858924390      instructions # 1.34  insn per cycle
```

```
# perf stat -- ./unpredictable_zicond

 Performance counter stats for './unpredictable_zicond':

  101149.07 msec task-clock
         97      page-faults
 1630745749      cycles
 4661493426      instructions # 2.86  insn per cycle
```

In misprediction demo application

(zicond with PGO)

+21% Instructions

1.77x Performance

2.13x IPC

In RISC-V Coremark (zicond extension w/o PGO)

1.07x Performance

Call for community PGO work

Summary

- Hardware sampling-based PGO is beneficial for:
  - Lower overhead
  - Higher accuracy
  - New feedback capabilities for higher performance gains

- Call community collaboration on RISC-V PGO to:
  - Widely adopt latest PMU extensions (CTR and more) in hardware
  - Stabilize infrastructure for RISC-V Kernel Perf framework
  - Propose more PMU features for more feedback types

- Contact me via Email: yichuan.gao@intel.com
- Special thanks to Intel SATG team for their invaluable work and support
  - More info: https://llvm.org/devmtg/2024-04/slides/TechnicalTalks/Xiao-EnablingHW-BasedPGO.pdf

RISE
RISC-V Software Ecosystem

- https://riseproject.dev

RISE is focused on positive and transparent collaborations with upstream projects to deliver commercial-ready software for various use cases

How: Align on highest priorities & avoid (accidental) duplication of work

Goal: Accelerate open source SW for RISC-V architecture

https://www.intel.com/content/www/us/en/developer/articles/community/rising-to-the-challenge-risc-v-software-readiness.html

Finding more interesting topics from Intel on RISC-V summit China 2024

Topic	**When & Where**
UXL 软件栈和 RISC-V 的初步探索	August 22 16:45 主会场A
LLVM 工具链 RISC-V 构建实现及其性能优化现状分析与未来展望	August 23 9:40 主会场A
GCC RVV 自动向量化及其应用	August 23 10:00 主会场A
Enhancing RISC-V Security with SBI Secure Service APIs	August 23 10:40 主会场B
Enabling Hardware Sampling Based PGO for RISC-V Platform	August 23 11:40 主会场A
利用 WASM 技术解决多种 ISA 的挑战	August 23 14:20 主会场B
HVP: Hardware Accelerated RISC-V Android Emulator	August 23 14:50 主会场A
Leverage BRS standard to improve RISC-V SW compatibility	August 23 17:30 主会场A
Soft-ISA: kernel built-in emulation engine to extend RISC-V silicon ISA capability	August 23 17:40 主会场A

Legal Notices and Disclaimers

intel®