Security Level: public

# Supporting RISC-V Full System Simulation in gem5

Prepared by Peter Yuen Ho Hin and Liao Xiongfei*

www.huawei.com

June 2021

*The full list of contributors can be found in paper

HUAWEI TECHNOLOGIES CO., LTD.

# Outline

- Introduction and Overview
- Target System
  - Hardware Configuration
  - Software Layers
  - Full System Linux Boot-up
- Challenges and Debugging Methodology
- Results
  - Diosix Boot-up
  - Benchmark Results

# RISC-V and gem5

- RISC-V ISA
  - free, open-source ISA, popular in both the academia and the industry
  - simple, efficient yet future-proof, adopts a modular approach

- gem5 Simulator
  - open-source simulator widely used in computer architecture research
  - configurable CPU models, memory sub-systems and peripherals
  - **Syscall Emulation (SE) mode**
    - effects of system calls are emulated
    - suitable for quickly running benchmarks in user mode
  - **Full System Simulation (FS) mode**
    - cycle-accurate simulation of a full-fledged system: OS + kernel, peripherals, interrupts etc.
    - required for accurate and realistic analysis of system performance

# RISC-V Full System Simulation in gem5

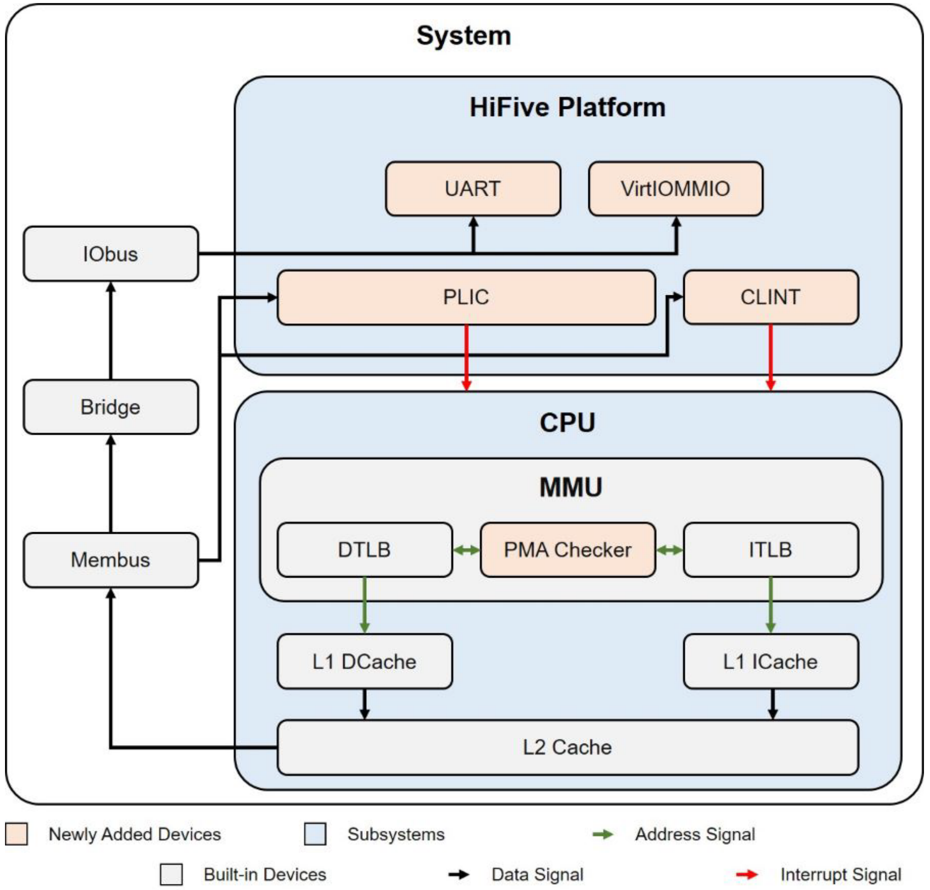
- Need for gem5 RISC-V Full System Simulation
  - enables more research possibilities: virtual memory, virtualization, distributed system, storage stack performance etc.
  - several online feature requests through gem5-users mailing list and stackoverflow
  - several projects within Huawei can benefit from gem5 RISC-V Full System simulation capability

- Existing RISC-V ISA Support in gem5
  - **“RISC5: Implementing the RISC-V ISA in gem5” @ CARRV 2017**: supporting most RISC-V instructions and system calls in SE mode
  - **“Simulating Multi-Core RISC-V Systems in gem5” @ CARRV 2018**: supporting thread-related system calls and synchronization instructions
  - **Our work**: adding Full System simulation capabilities to gem5 RISC-V by developing peripheral models, fixing privileged instructions and interrupt handling as well as providing preliminary hypervisor support

- source code is available in official **gem5-21.0 release** (March 2021)

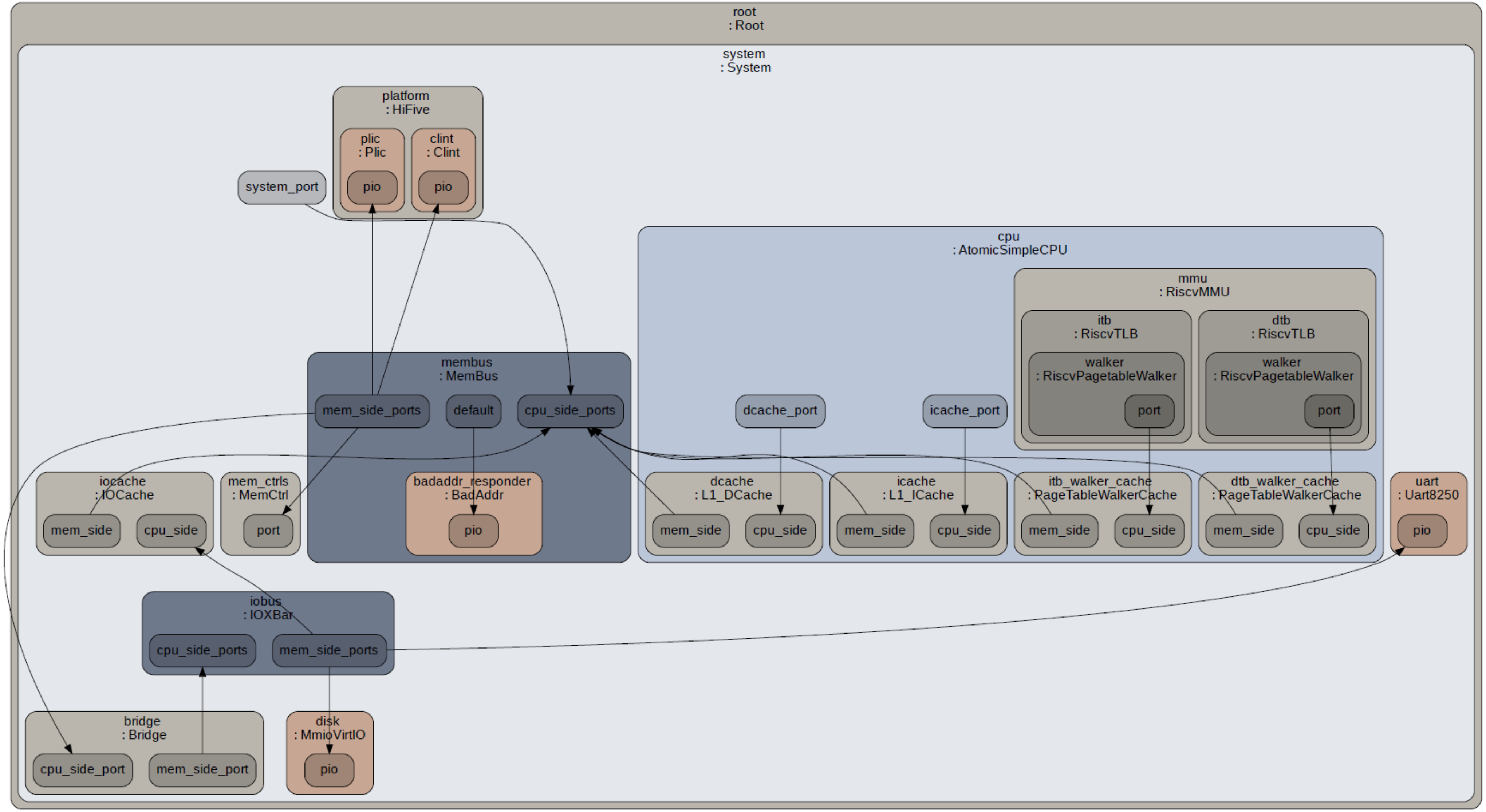
# Hardware Configuration of Target System

Target System: a baseline RISC-V system which can be easily extended based on user needs




- CPU Sub-system
  - PMA checker (extra MMU component)
    - checking physical memory attributes such as atomicity, memory-ordering, coherence, **cacheability** and idempotency
- HiFive Platform
  - based on SiFive's HiFive series of board
  - **Core Local Interrupter (CLINT)**: handles software and timer interrupts via an MMIO interface
  - **Platform Level Interrupt Controller (PLIC)**: routing external interrupts to the hardware threads based on a priority scheme
  - **UART**: provides an interactive command line terminal
  - **VirtIOMMIO**: provides a copy-on-write root filesystem which contains the workload scripts and operating system binaries

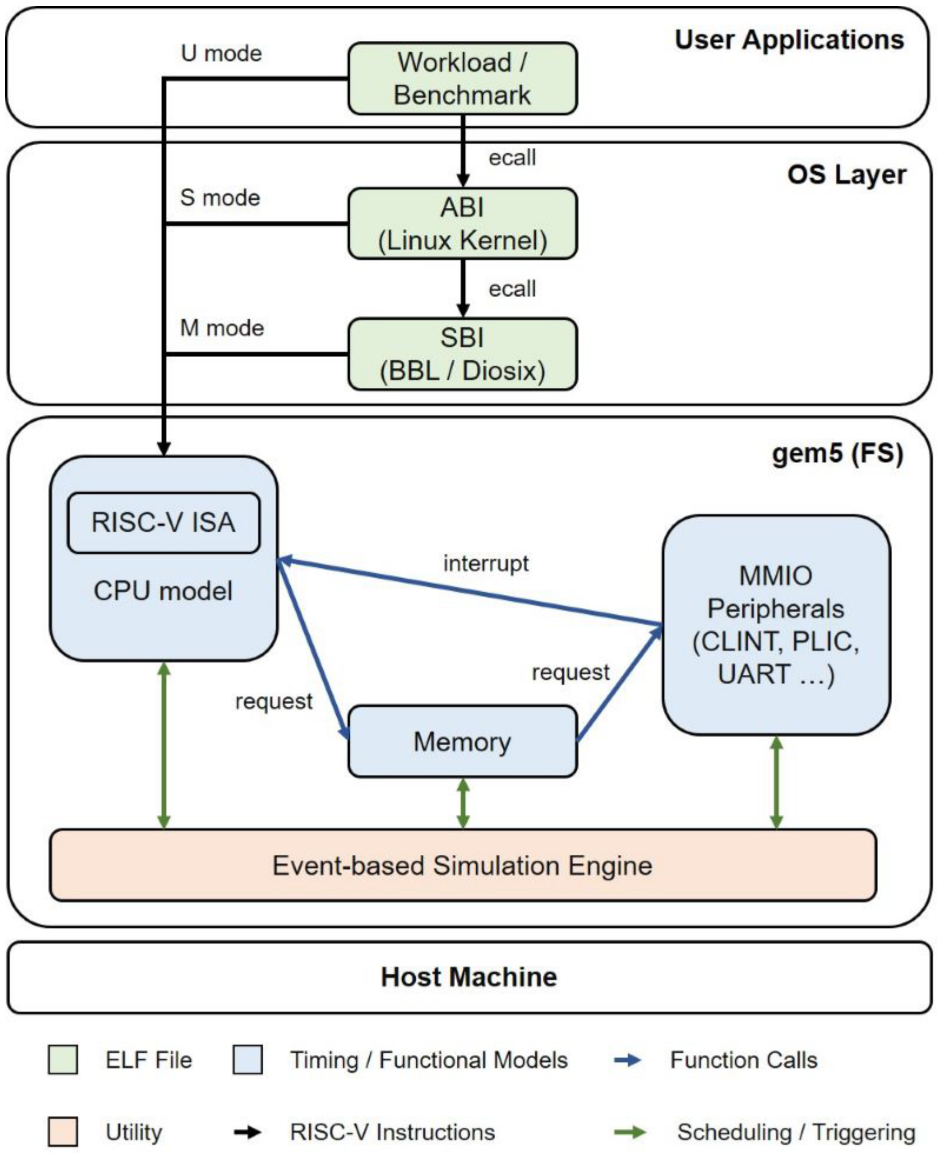
# Hardware Configuration in gem5



# Software Layers of Target System

- gem5 (FS) Block
  - simulates functional and timing behaviour of hardware devices using an event-based approach
  - parses ELF workloads and sends machine instructions to CPU model

- CPU Model with RISC-V ISA Decoder
  - decodes the machine instructions and updates register state / performs memory requests



# Full System Linux Boot-Up

- Setup
  - SMP with 4 CPU cores, each with one hardware thread
  - Berkeley bootloader (bbl) + Linux kernel v5.10
  - Filesystem: BusyBox disk image with PARSEC benchmark

- Features
  - Supports all four CPU models: Atomic Simple, Timing Simple, Minor and DerivO3
  - Multi-threaded workloads
  - Checkpoint and restoration functionalities for switching CPU models
  - m5 pseudo-instructions for dumping benchmark statistics / checkpoints via the terminal
  - Fully functional filesystem
  - Easy configuration through Python (e.g. automatic DTB generation)

```
[    0.311007] virtio_blk virtio0: [vda] 524288 512-byte logical blocks (268 MB/256 MiB)
[    0.311116] vda: detected capacity change from 0 to 268435456
[    0.317180] libphy: Fixed MDIO Bus: probed
[    0.319758] e1000e: Intel(R) PRO/1000 Network Driver
[    0.319827] e1000e: Copyright(c) 1999 - 2015 Intel Corporation.
[    0.320173] ehci_hcd: USB 2.0 'Enhanced' Host Controller (EHCI) Driver
[    0.320256] ehci-pci: EHCI PCI platform driver
[    0.320423] ehci-platform: EHCI generic platform driver
[    0.320640] ohci_hcd: USB 1.1 'Open' Host Controller (OHCI) Driver
[    0.320768] ohci-pci: OHCI PCI platform driver
[    0.320935] ohci-platform: OHCI generic platform driver
[    0.321827] usbcore: registered new interface driver uas
[    0.322055] usbcore: registered new interface driver usb-storage
[    0.322445] mousedev: PS/2 mouse device common for all mice
[    0.324000] usbcore: registered new interface driver usbhid
[    0.324073] usbhid: USB HID core driver
[    0.326590] NET: Registered protocol family 10
[    0.328790] Segment Routing with IPv6
[    0.329065] sit: IPv6, IPv4 and MPLS over IPv4 tunneling driver
[    0.330865] NET: Registered protocol family 17
[    0.331615] 9pnet: Installing 9P2000 support
[    0.331828] Key type dns_resolver registered
[    0.332146] debug_vm_pgtable: [debug_vm_pgtable         ]: Validating architecture page table hel
pers
[    0.334822] EXT4-fs (vda): mounting ext2 file system using the ext4 subsystem
[    0.338688] EXT4-fs (vda): mounted filesystem without journal. Opts: (null)
[    0.338844] VFS: Mounted root (ext2 filesystem) readonly on device 254:0.
[    0.340905] devtmpfs: mounted
[    0.341456] Freeing unused kernel memory: 220K
[    0.345704] Run /sbin/init as init process
ip: can't find device 'eth0'
ip: SIOCGIFFLAGS: No such device
ip: can't find device 'eth0'

Welcome to RiscV

UCanLinux login:
```

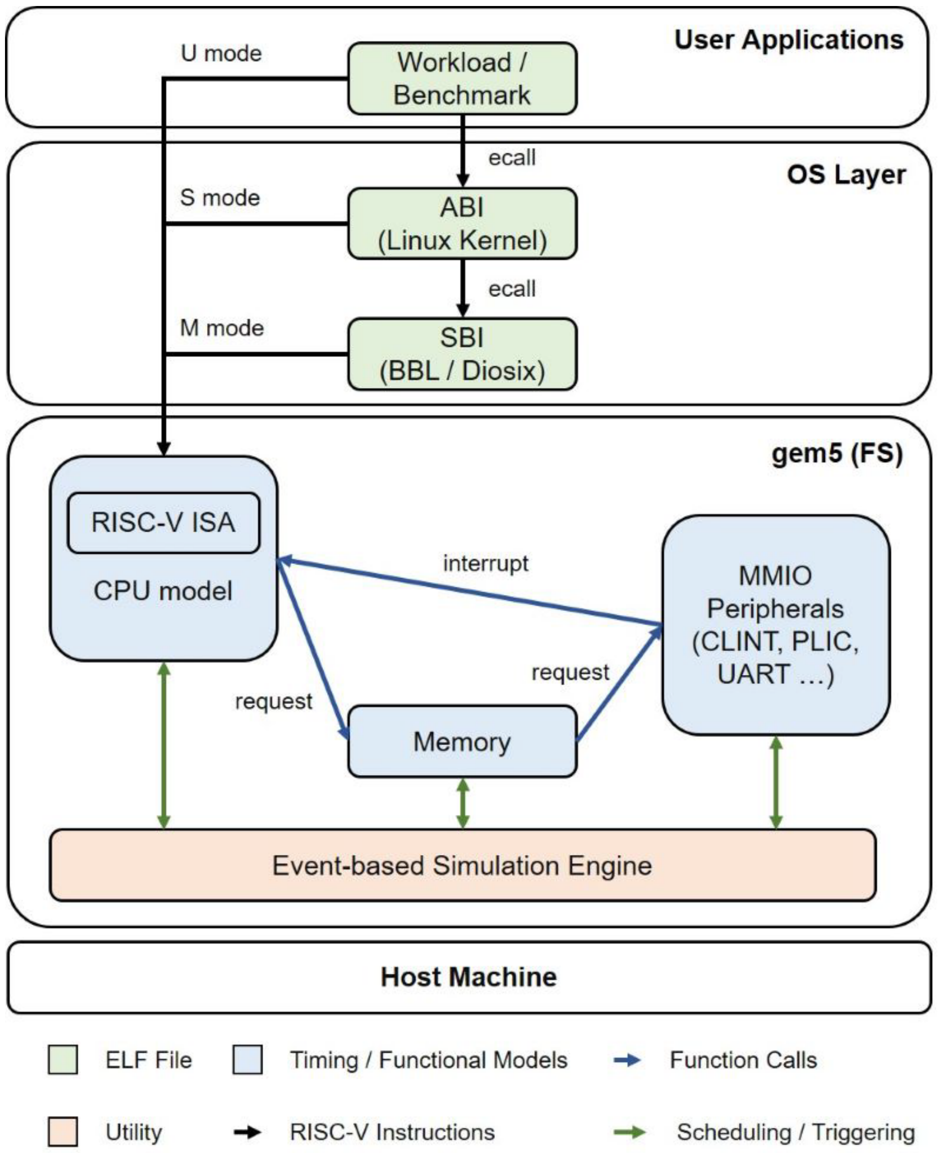
```
UCanLinux login: root
root
Password: root

root@UCanLinux:~ #
root@UCanLinux:~ # ls
ls
index.html
```

```
root@UCanLinux:~ # echo "Hello World"
echo "Hello World"
Hello World
```

# Challenges

- A fault can occur due to numerous reasons
  - DTB configuration error
  - wrong privileged ISA implementation
  - interrupt triggering mechanism or interrupt handling logic within CPU and interrupt controller
  - errors due to CPU pipeline and memory access of peripheral devices etc.

- Debugging involves 2 parts
  - kernel payload (in RISC-V assembly language, debugged using remote GDB)
  - gem5 implementation (in C++, debugged using host GDB)



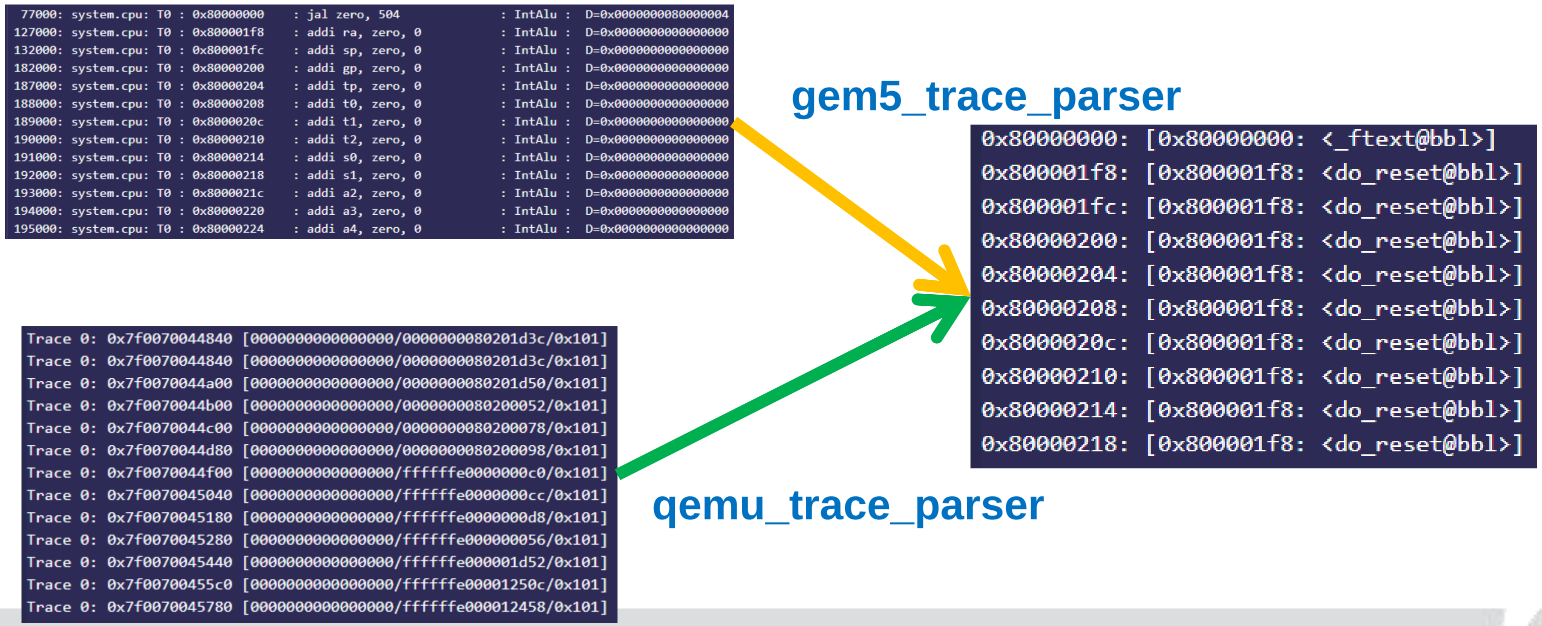
# Challenges (cont.)

- **Kernel payload:** difficult to identify the instruction triggering the fault
  - large code size of kernel / bootloader
  - debugging mostly in assembly language
  - errors in kernel ends up in an infinite loop instead of exiting with an error
  - simple remote GDB is not suitable

- **gem5 implementation**: event-driven nature complicates debugging process
  - call-stack information is limited to calls within the same simulation tick
  - events such as memory read request and response would not be visible under the same call stack
  - a method of analysing beyond the scope of current tick is needed
  - relied heavily on debug logging (inefficient due to frequent rebuilding of gem5 binary)

# Debugging Methodology

- Enhanced remote GDB support for RISC-V in gem5
  - added support for getting and setting values of FP and CSR registers for checking of privileged instruction implementation and interrupt register states
- Trace analysis and debugging using Python toolkit
  - boot QEMU and gem5 FS side-by-side using the same system setup and collect both execution traces
  - parse both traces and perform comparisons on the execution paths to find out where two traces diverge
  - allows for programmatic control of remote GDB instance (inserting breakpoints, comparing register states etc.)



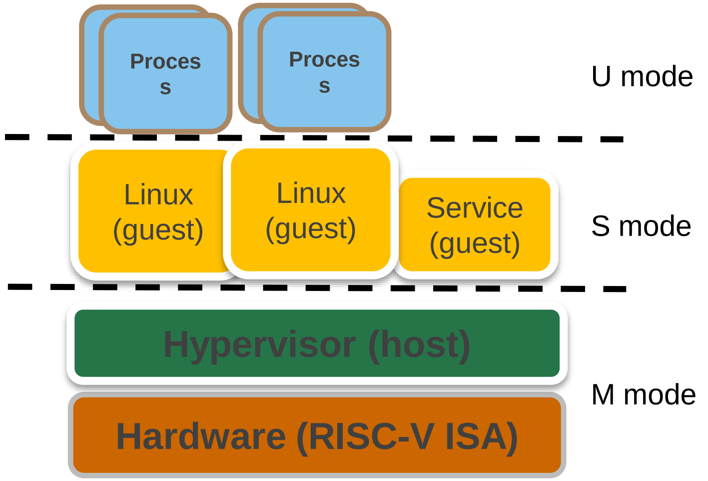
# Debugging Process

- Incorrect return value from a load instruction
  - incorrect implementation of a peripheral device
  - accidentally caching of the MMIO address range

- Interrupt trigger mechanism
  - requires both logging and remote GDB
  - remote GDB
    - examine the CPU’s interrupt handling logic (e.g. Minor CPU repeatedly enters interrupt handler function)
  - logging
    - examine internal state of gem5 device models (e.g. PLIC’s input and output registers)
    - investigate the sequence of events (verify timing behaviour of interrupt controllers)

- gem5’s built-in debug logs
  - inspect detailed activities of built-in device models (CPU models, TLBs, MMU etc.)
  - identified an issue of CLINT’s RTC accidentally triggering squashes in DerivO3CPU

# Diosix Boot-up



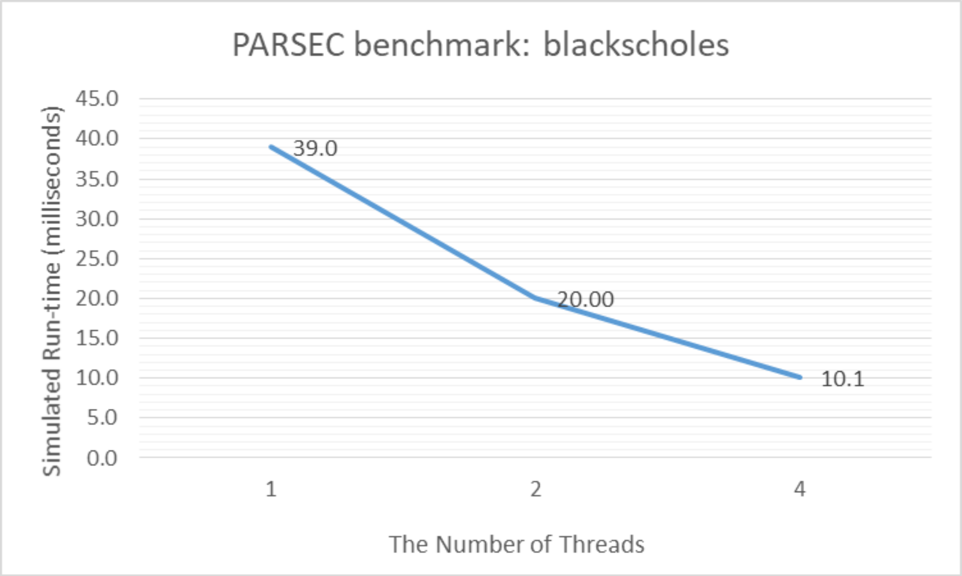

- Diosix
  - an M mode hypervisor
  - can be used to perform hypervisor studies before H mode support is available

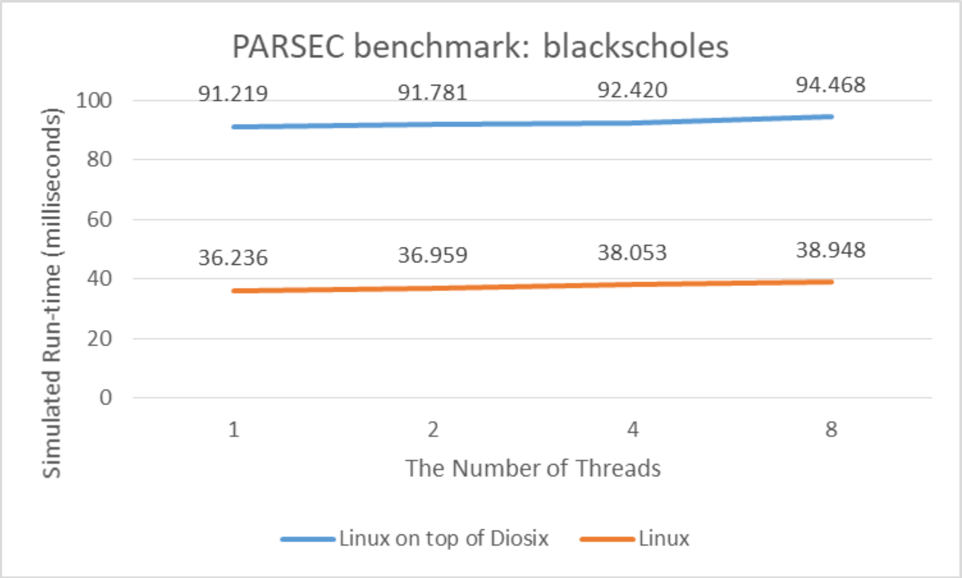
- System configuration of gem5 for bringing up DioSix
  - Using full system mode in gem5
    - 2 guest kernels (Linux kernel + UART service)
  - Using Linux Kernel 5.8
    - 5.10 has jump table bugs during kernel start-up (self-patching)

- Result
  - all guest are scheduling and running to idle
  - benchmarks can be run in the guest OS
  - interactive terminal using system service guest

# Benchmark Results

- PARSEC blackscholes benchmark
  - FS simulation
    - simulated run-time scales well with the number of threads
    - 4 CPU cores, each with one hardware thread
  - Linux vs Linux on top of Diosix hypervisor
    - check the overheads due to hypervisor
    - 1 CPU core with one hardware thread



# Thank you

www.huawei.com