



SMP-Difftest

支持多处理器的差分测试方法

王凯帆 王华强
中科院计算技术研究所
2021年6月24日

目录

- 使用指令级在线仿真验证框架 差分测试 (*difftest*) 验证处理器
- 通过微结构状态对齐实现在线差分测试框架
- 差分测试框架的 SMP (对称多处理器) 拓展

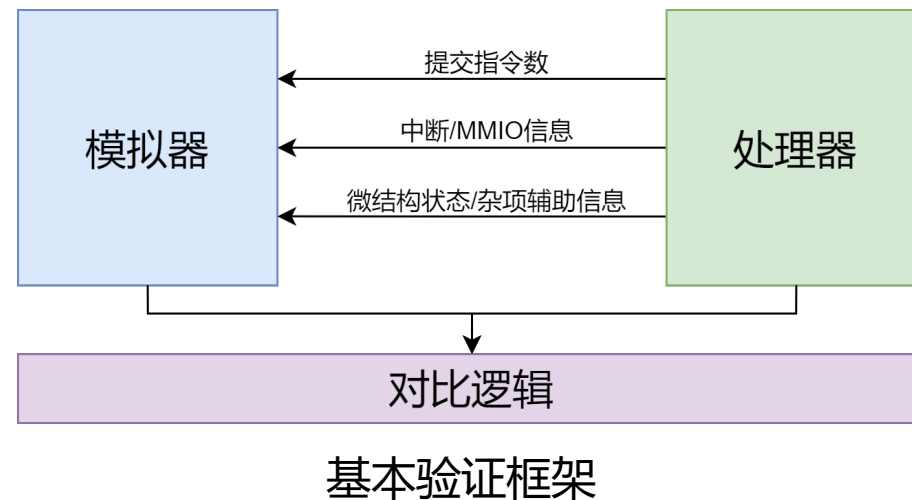
传统处理器整核验证的问题

- 传统的整核验证:
 - 生成正确的 Trace 后运行处理器并对比 Trace

	传统测试方法
灵活性	低 <ul style="list-style-type: none">• 需要提前生成Trace• 无法应对变化的外部输入 (如外部设备状态的变化)• 无法处理多核的情况
存储空间占用	<ul style="list-style-type: none">• 预先生成的Trace需要占用空间

Difftest_[1] 机制

- 香山的全系统仿真验证使用 *Difftest* 机制
- 指令级的在线仿真验证框架
- 执行流程
 - (1) 处理器仿真产生指令提交
 - (2) 模拟器执行相同的指令
 - (3) 比较两者状态



```
while (1) {  
    icnt = cpu_step();           //(1)  
    nemu_step(icnt);             //(2) [2]  
    r1s = cpu_getregs();         //(3)  
    r2s = nemu_getregs();        //(3)  
    if (r1s != r2s) { abort(); } //(3)  
}
```

在线验证机制

[1] Yu, EasyDiff: An Effective and Efficient Framework for Processor Verification, CRVF 2019, <https://crvf2019.github.io/pdf/14.pdf>

[2] NJU Emulator (NEMU)是南京大学开发的轻量级教学用模拟器. 香山在验证中使用其作为模拟器.

Difftest 机制的性能表现

- 模拟器的速度比仿真速度快几个数量级

程序输入/每秒指令数	模拟器	处理器仿真	接入Difftest后 处理器仿真
CoreMark	1079948	7543	7503

- 通过**动态链接**接入 Difftest 机制, 通信开销小
- Difftest 对仿真的速度的影响**微乎其微**

仅依靠模拟器验证的瓶颈

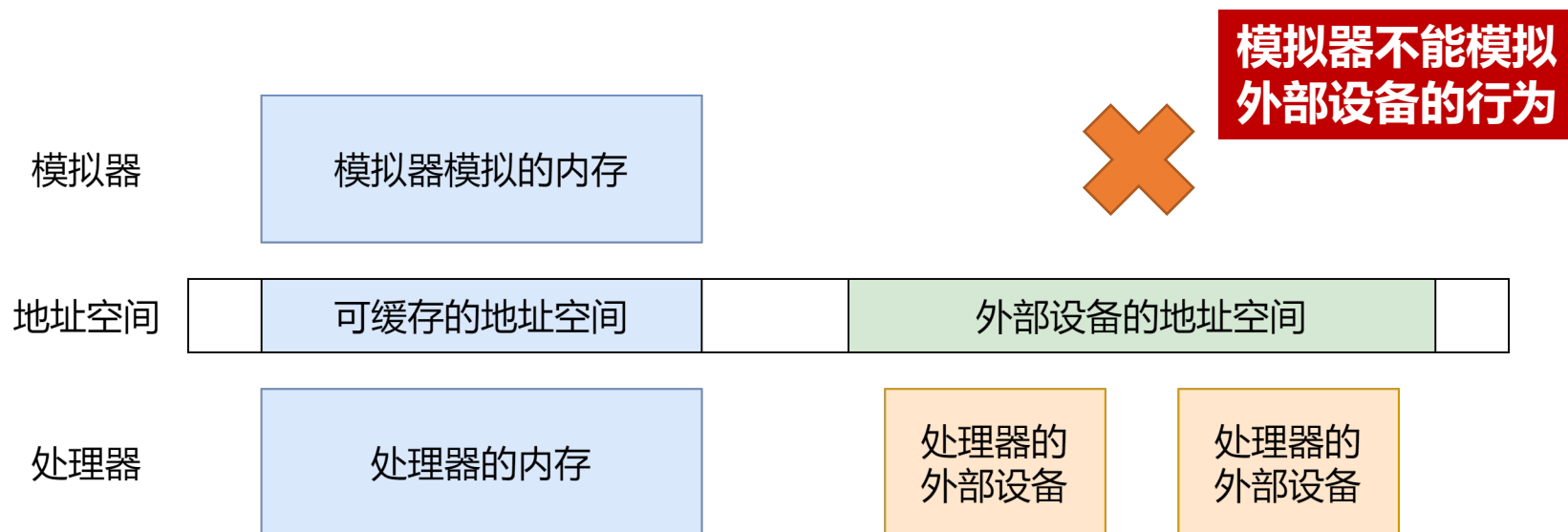
- 模拟器无法仅靠自己在一些行为上与正确的处理器对齐
- 无法依靠模拟器直接验证处理器的行为

与外部输入相关的行为	与微结构相关的行为	与一致性相关的行为
外部中断	时钟中断	LR/SC
MMIO	Store Page Fault	多核



无法对齐的行为分析

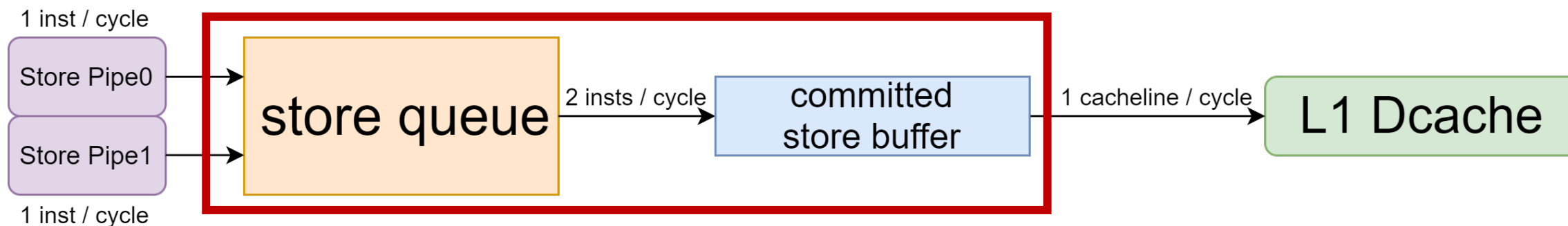
- Memory Mapped IO (MMIO)
 - 映射到内存地址的IO指令
 - 表现为访问特定地址区域的 load / store 指令
 - 模拟器**不能模拟**所有的外部设备
 - 模拟器无法得知这些 load 指令的正确结果





无法对齐的行为分析

- Page Fault
 - Page Fault 的产生与否可能与微结构有关



会存储一部分已提交 store 的数据
这些数据对页表是**不可见**的

- 在**没有**使用同步指令的情况下，对**页表**的更改是否对地址映射产生影响？
 - RISC-V 标准没有给出显式的描述
 - 这意味着是否产生影响**均是可以接受**的行为

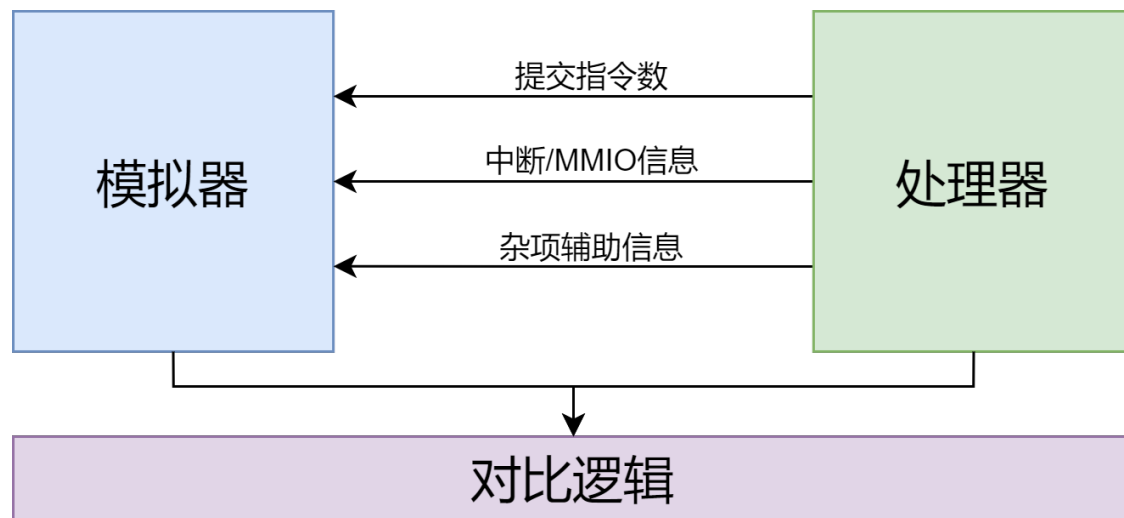
解决方案

- 上述问题的原因
 - 模拟器缺乏微结构信息
- 解决方案
 - 传递微结构状态，以同步模拟器与处理器
- 问题
 - **传递什么**微结构状态？
 - 模拟器**如何检查**传入的微结构状态是否合法？
 - 如何根据传入的状态**更新**模拟器状态？



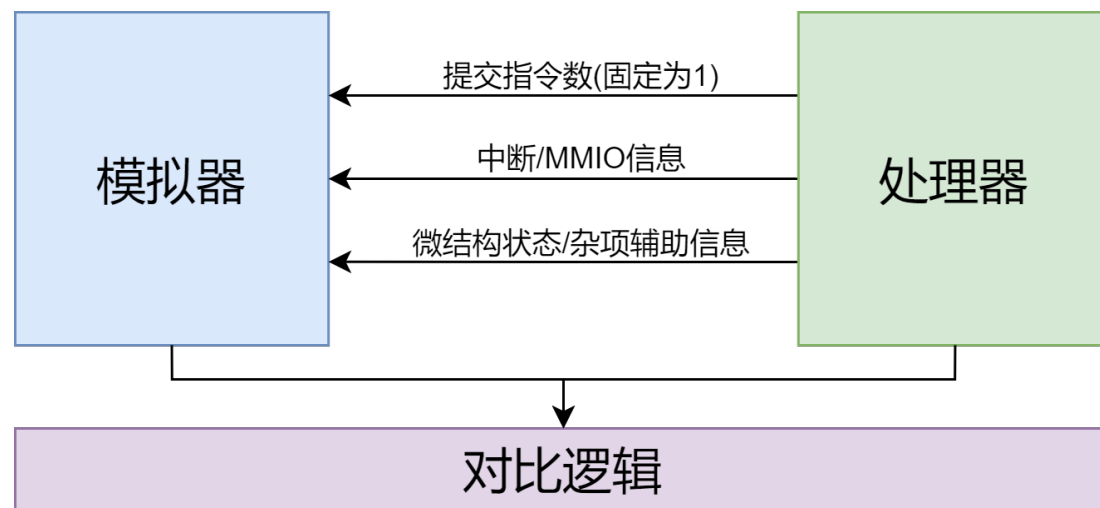
两种比对方式(1): 不检查传入状态

- exec() : 正常执行
 - 会将处理器触发的**中断信息、MMIO结果**传递给模拟器
 - 模拟器认为处理器中断的触发、MMIO 读取的数据是**全部正确的**
 - 模拟器**不会检查**传入的信息的正确性
 - 中断处理是否正确仍会被检查
 - 用于正常执行、中断提交、MMIO提交



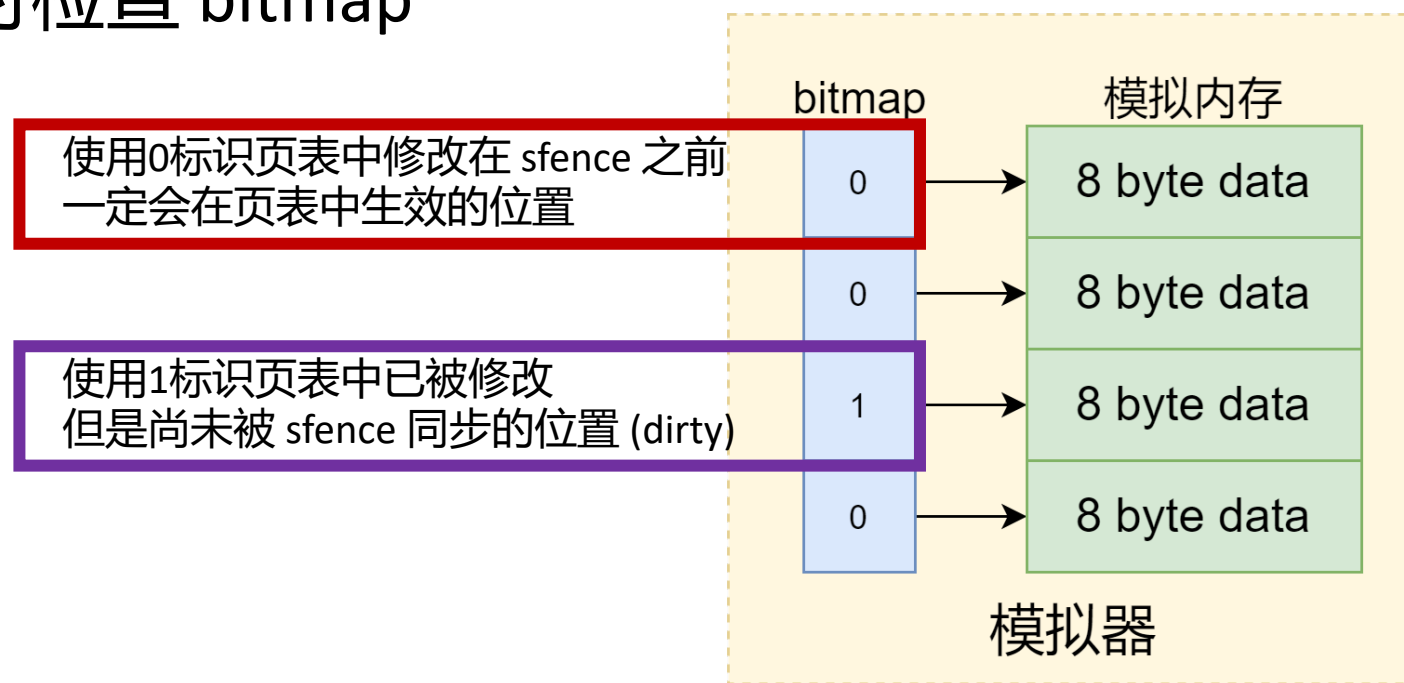
两种比对方式(2): 检查传入状态

- disambiguate_exec(): “消歧义执行”
 - 根据传入微结构状态, 在对应节点改变模拟器状态
 - 模拟器**检查**强制状态变动是否合法, 合法后才执行
 - 在出现 lr/sc, page fault 时使用
- LR/SC
 - 仅允许成功->失败的**单向**变动
- Page Fault



是否要严格追踪页表的状态?

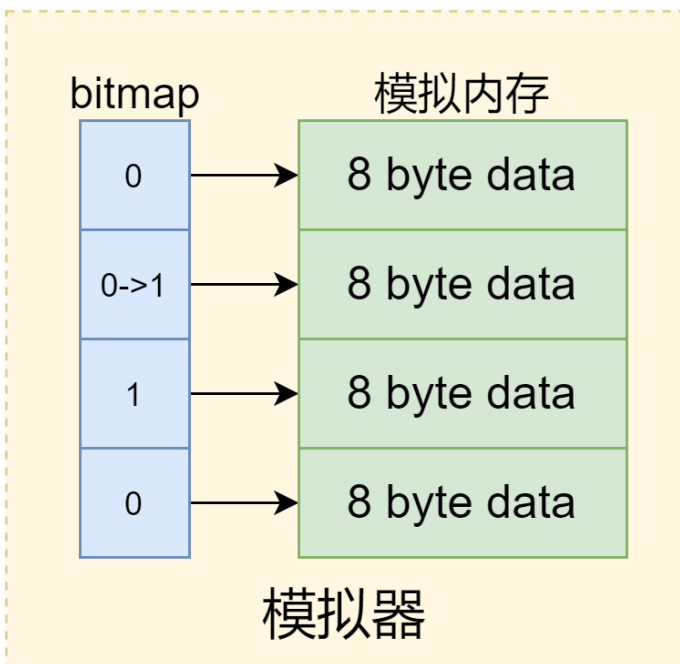
- 判断 Page Fault 是否合法时，需要检查对页表的写入是否已经被 sfence 过
- 可否使用 **bitmap** 对页表的状态进行追踪？
- 在模拟器读取页表时检查 bitmap



是否要严格追踪页表的状态?

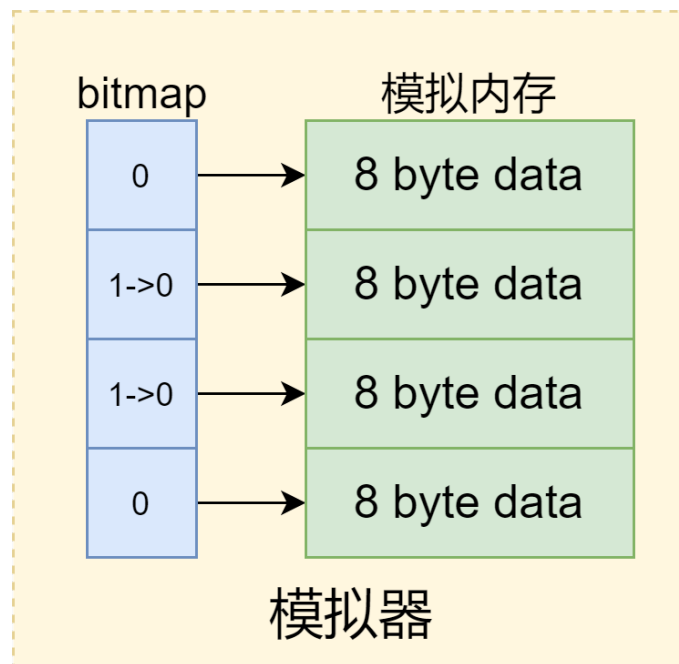
如果使用 **bitmap** 对页表的状态追踪:

store 写内存



将被写入的位置设置为dirty状态

sfence



清除所有的dirty状态

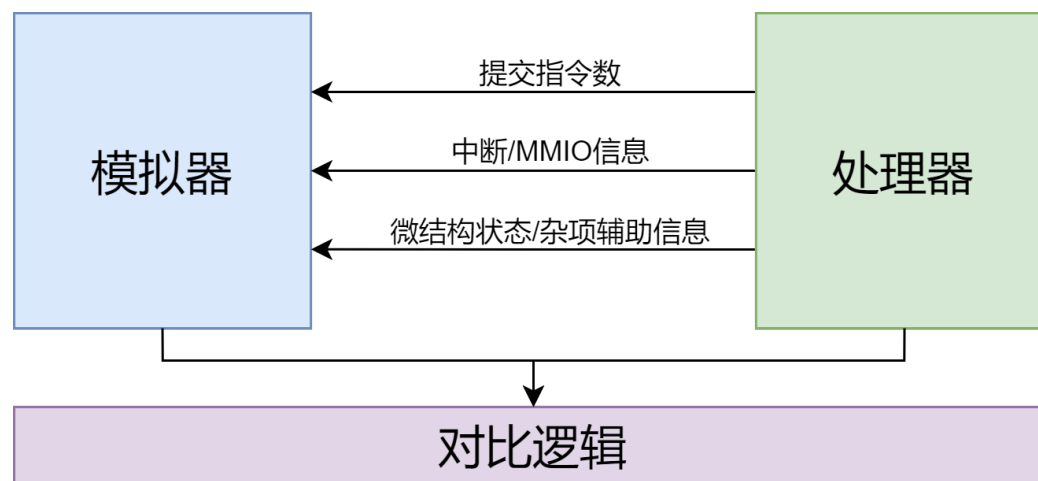
是否要严格追踪页表的状态?

- 否
- 会引起模拟器的性能问题
- 模拟器会在每次 sfence 时刷新整个 bitmap
 - 以内存大小128MB为例
 - Bitmap **以 byte 为粒度** -> bitmap 大小为16MB
 - 降低模拟器的执行速度
 - **后果: 模拟器做 sfence 的开销很大**
 - 解决方案: 降低 bitmap 的追踪粒度 / 不严格追踪页表的状态 / 限制追踪的范围



小结：通过微结构状态同步实现协同仿真

- 处理器向模拟器传递微结构状态
- 模拟器检查处理器传来的状态，调整自身状态
- 将处理器与模拟器的执行结果比对



- 实现**单核**下处理器与模拟器的协同仿真

仿真验证手段的缺陷

- 提升仿真验证效率和覆盖面是敏捷开发的关键一环
- 目前，香山处理器相关基础设施已经较为完备
 - 但是缺少 **SMP 结构**下的**全系统**仿真验证手段

验证粒度 \ 系统结构	系统结构	
	单核结构	多核结构
功能部件	RV-Test	TL-C VIP
全系统	Difftest	? ? ?

SMP 仿真验证面临的挑战

① 核间中断

CLINT、PLIC

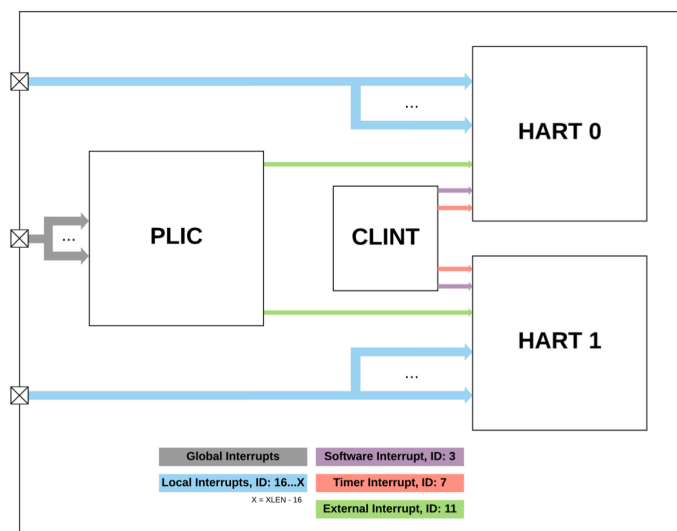
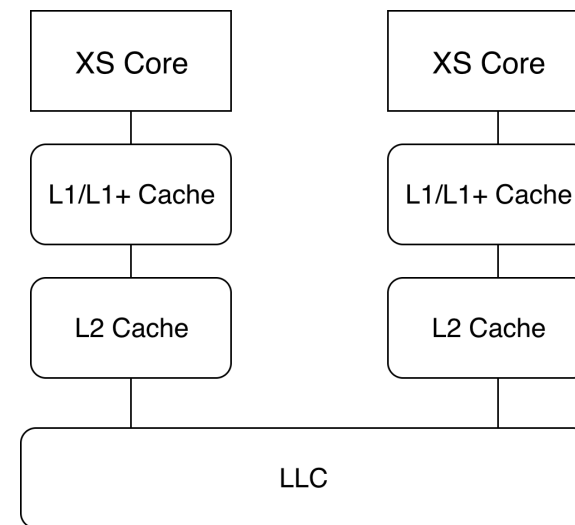


Figure 8: PLIC + CLINT PLIC Block Diagram for Machine Mode

RV64 中断控制器模型*

② 内存一致性同步

Cache Coherence & Memory Consistency




香山处理器 SMP 模型

* From SiFive Interrupt Cookbook V1.0

SMP Difftest

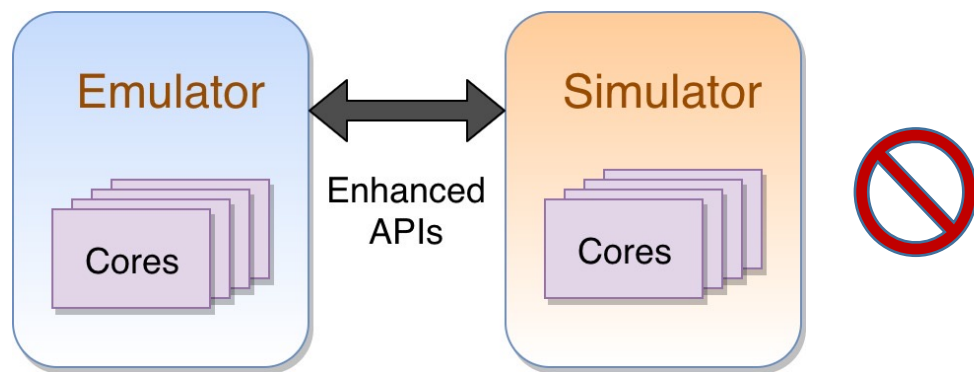
- 支持 SMP 结构的全系统仿真验证框架
 - 继承自 Difftest 框架，支持多核情景下的全系统仿真验证
- **拓宽验证集**：支持多线程程序、SMP Linux 内核等 workload
- **提升验证力**：支持检测内存一致性方面的软硬件问题

验证粒度 \ 系统结构	系统结构	
	单核结构	多核结构
功能部件	RV-Test	TL-C VIP
全系统	Difftest	



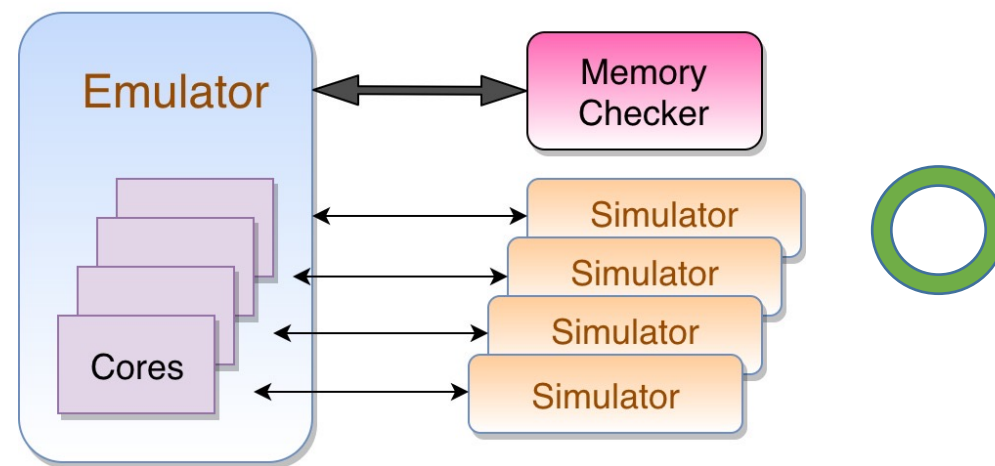
两种拓扑选择

① 模拟器扩展到多核，整体相连



工程量较大
灵活性不足
难以做到两者缓存之间的同步

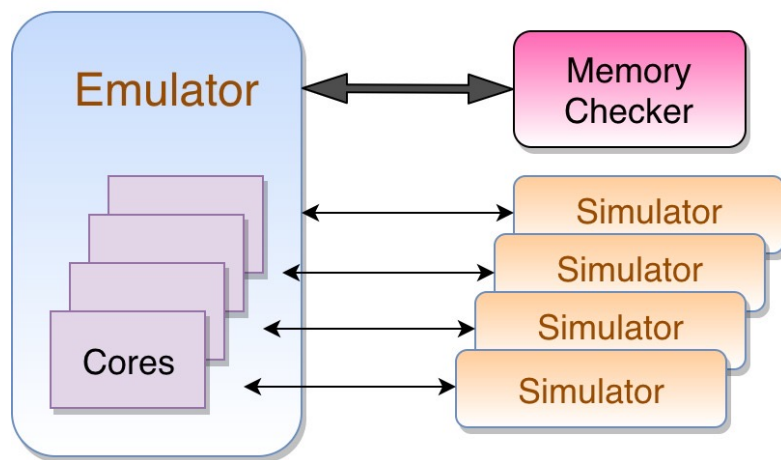
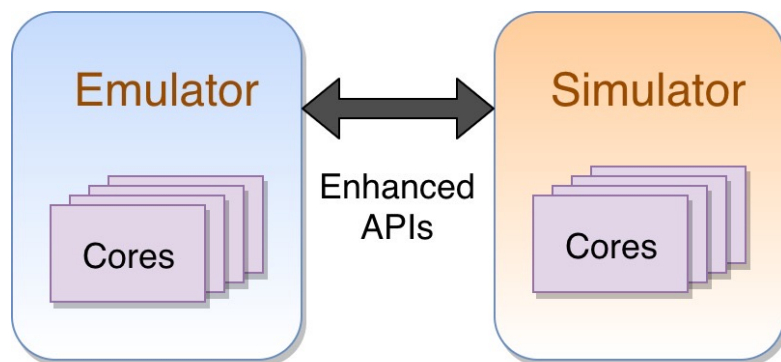
② 与模拟器一对一连接



工程量较小
解耦模块以便调试
便于验证缓存一致性

VS.

整体逻辑架构

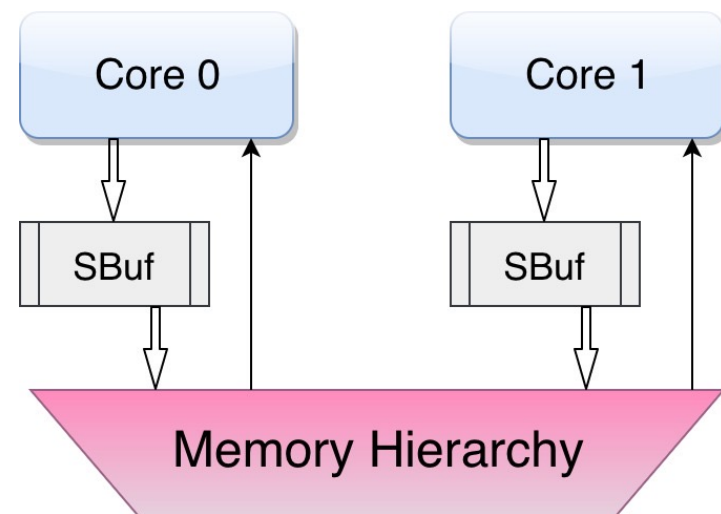


```
while (1) {  
    for (i <= 0 .. NrCores) {  
        verilator_step(i);  
        nemu_step(i);  
        verilator_getregs(i,&r1);  
        nemu_getregs(i,&r2);  
        if (r1 != r2) abort();  
        if (memory_checker failed) abort();  
    }  
}
```

Memory Checker is the **Key** !

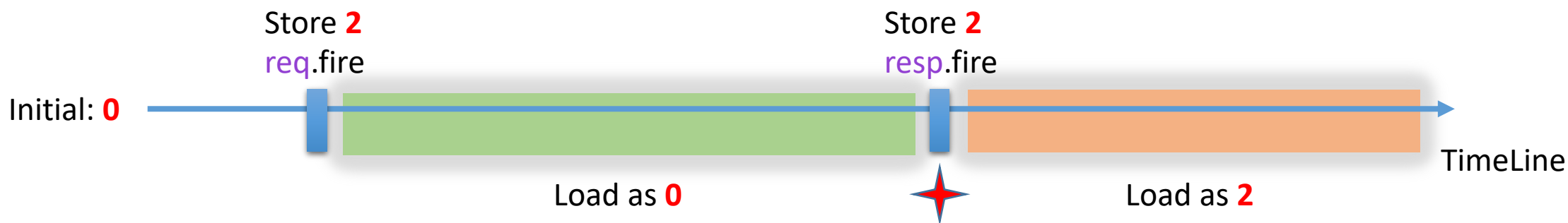
Cache Coherence 验证

- 假设一般处理器访存模型简化为右图：
- 重点在于验证以下两条机制
 - 写传播 (write propagation)
 - 事务串行化 (transaction serialization)
- 核心问题
 - Store 指令有无可线性化点？如何确定可线性化点？



Cache Coherence 验证

- 结合前述设定与 TL-C 一致性协议的要求，有以下论断：
 - 核 i SBuf 接收到 D\$ 返回信号之前，核 j 必~~不会~~读取返回核 i 写入的**新值**
 - 核 i Sbuf 接收到 D\$ 返回信号之后，核 j 必~~不会~~读取返回核 i 写入之前的**旧值**
- 结论：
 - SBuf 收到 D\$ resp 信号的那一刻为内存写生效的**可线性化点**



Cache Coherence 验证

- 基于上述结论，在仿真框架中增加 Global-Memory 部件
 - 用于维护全局 Memory Hierarchy 状态
 - 使用微结构状态更新 Global-Memory

微结构事件	更新内容
SBuf 写请求收到 resp 信号	Store 进入全局访存历史
Atomic Unit 收到 resp 信号	原子指令进入全局访存历史

- 当 Load 从 D\$ refill 数据时，检查其是否与 Global-Memory 一致
 - 如果**不一致**，判定该访存违反 Cache Coherence 原则，现场报错

Memory Consistency 验证

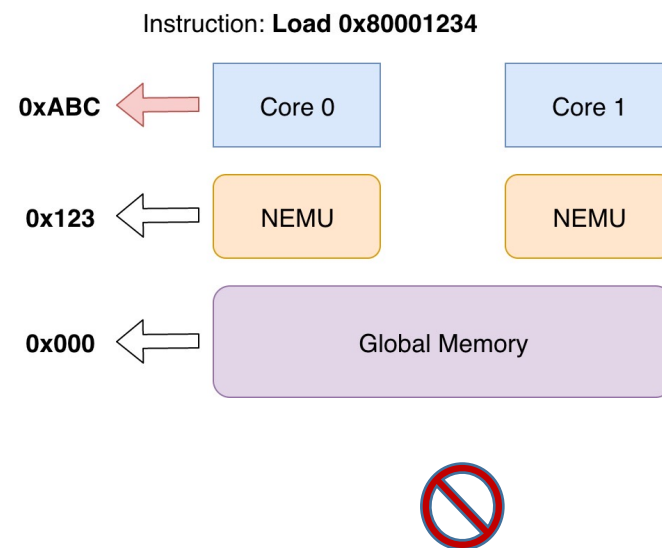
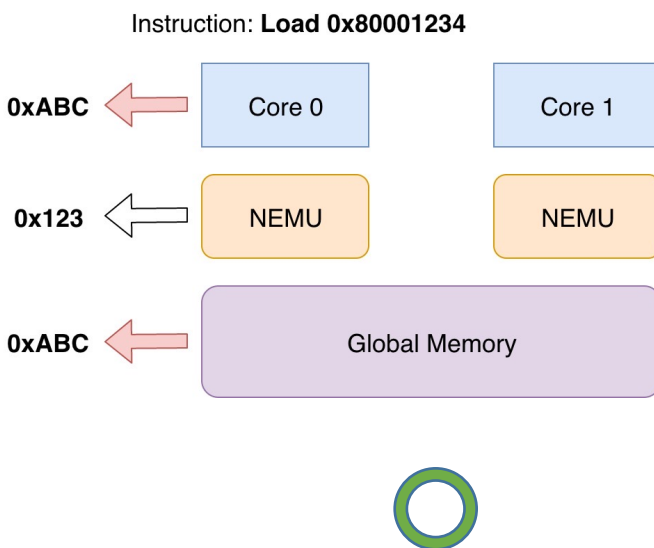
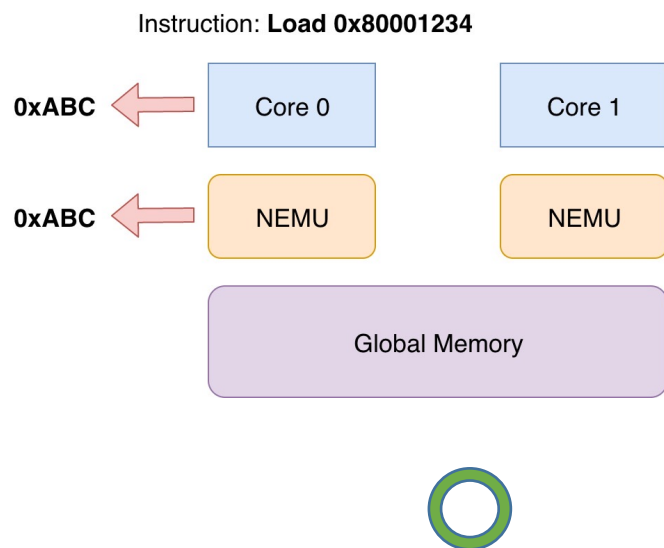
- 内存模型：RISC-V WEAK MEMORY ORDERING (RVWMO)
 - Global Memory Order
 - Preserved Program Order (PPO)
 - Load Value Axiom
- Commit 阶段利用 Global-Memory 尽可能检查各种 Axioms



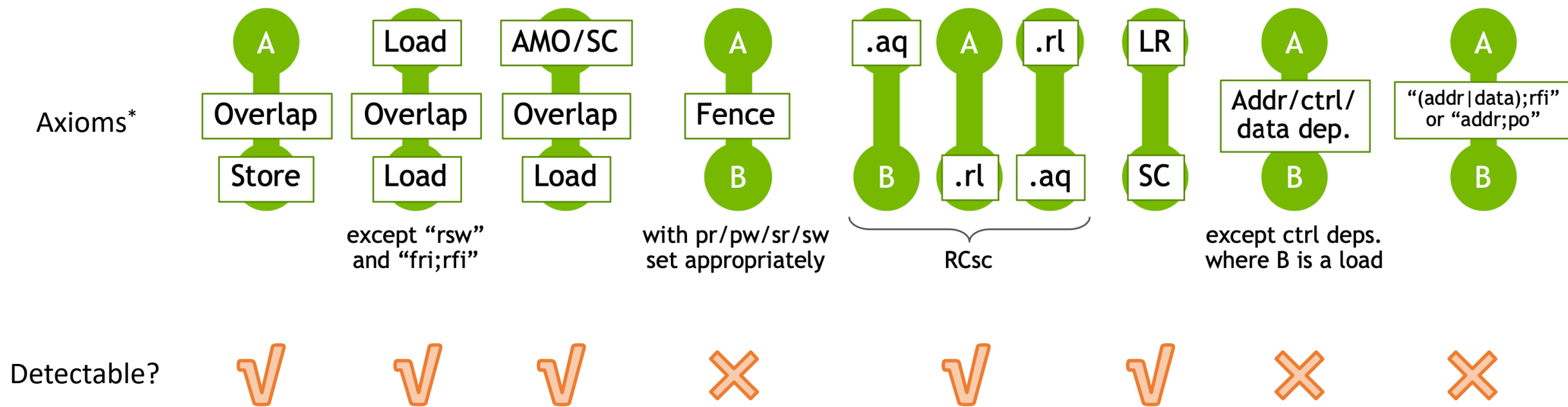


Load Value Axiom

- Load 的返回值可能遵循两种定序，一个是在本地的程序序，另一个是全局访存序（以较晚者为准）
- 全局访存序的结果由 Global-Memory 提供
- 本地程序序的结果由 NEMU 模拟器提供



Preserved Program Order



* From RISC-V Memory Consistency Model Tutorial

总结

• 应用效果

- 拓宽验证集种类
- 提升已有验证集覆盖
- 发现软硬件相关 Bug

Cache 预取状态转移错误
mip Dfftest 状态不一致
原子访问外设未支持

.....

```
[ 0.000000] OF: fdt: Ignoring memory range 0x80000000 - 0x80200000
[ 0.000000] Linux version 4.18.0-00048-g9be229d2ec2c-dirty (wkf@xiangshan-06) (gcc version 9.2.0 (GCC)) #159 SMP Sur
[ 0.000000] bootconsole [early0] enabled
[ 0.000000] Initial ramdisk at: 0x(____ptrval____) (23552 bytes)
[ 0.000000] Zone ranges:
[ 0.000000]   DMA32     empty
[ 0.000000]   Normal [mem 0x0000000080200000-0x0000000081ffffff]
[ 0.000000] Movable zone start for each node
[ 0.000000] Early memory node ranges
[ 0.000000]   node    0: [mem 0x0000000080200000-0x0000000081ffffff]
[ 0.000000] Initmem setup node 0 [mem 0x0000000080200000-0x0000000081ffffff]
[ 0.000000] Cannot allocate SWIOTLB buffer
[ 0.000000] elf_hwcap is 0x112d
[ 0.000000] percpu: Embedded 11 pages/cpu @(__ptrval__) s15072 r0 d29984 u45056
[ 0.000000] Built 1 zonelists, mobility grouping on.  Total pages: 7575
[ 0.000000] Kernel command line: root=/dev/mmcblk0 rootfstype=ext4 ro rootwait earlycon
[ 0.000000] Dentry cache hash table entries: 4096 (order: 3, 32768 bytes)
[ 0.000000] Inode-cache hash table entries: 2048 (order: 2, 16384 bytes)
[ 0.000000] Sorting __ex_table...
[ 0.000000] Memory: 28936K/30720K available (780K kernel code, 78K rdata, 109K rodata, 110K init, 100K bss, 1784K )
[ 0.000000] SLUB: HWalign=64, Order=0-3, MinObjects=0, CPUs=2, Nodes=1
[ 0.000000] Hierarchical RCU implementation.
[ 0.000000] NR_IRQS: 0, nr_irqs: 0, preallocated irqs: 0
[ 0.000000] clocksource: riscv_clocksource: mask: 0xffffffffffffffff max_cycles: 0x1d854df40, max_idle_ns: 35263616
[ 0.000000] console [hvc0] enabled
[ 0.000000] console [hvc0] enabled
[ 0.000000] bootconsole [early0] disabled
[ 0.000000] bootconsole [early0] disabled
[ 0.000000] Calibrating delay loop (skipped), value calculated using timer frequency.. 2.00 BogoMIPS (lpj=10000)
[ 0.000000] pid_max: default: 4096 minimum: 301
[ 0.000000] Mount-cache hash table entries: 512 (order: 0, 4096 bytes)
[ 0.000000] Mountpoint-cache hash table entries: 512 (order: 0, 4096 bytes)
[ 0.000000] Hierarchical SRCU implementation.
[ 0.000000] smp: Bringing up secondary CPUs ...
[ 0.000000] smp: Brought up 1 node, 2 CPUs
[ 0.000000] clocksource: jiffies: mask: 0xffffffff max_cycles: 0xffffffff, max_idle_ns: 19112604462750000 ns
[ 0.000000] clocksource: Switched to clocksource riscv_clocksource
[ 0.000000] Unpacking initramfs...
[ 0.000000] workingset: timestamp_bits=62 max_order=13 bucket_order=0
[ 0.000000] random: get_random_bytes called from 0xffffffff8001f912 with crng_init=0
[ 0.000000] Freeing unused kernel memory: 108K
[ 0.000000] This architecture does not have kernel memory protection.
Hello, RISC-V World!
HIT GOOD TRAP at pc = 0x7f80034ee6
total guest instructions = 5,626,704
```

SMP Linux Kernel 启动

Overhead: <1%

**谢谢！
请批评指正**
